RECURSIVE DEFINITIONS OF PARTIAL FUNCTIONS AND THEIR COMPUTATIONS

J. M. Cadiou

Stanford University

Prepared for:

Advanced Research Projects Agency

March 1972

DISCLAIMER NOTICE

THIS DOCUMENT IS THE BEST QUALITY AVAILABLE.

COPY FURNISHED CONTAINED A SIGNIFICANT NUMBER OF PAGES WHICH DO NOT REPRODUCE LEGIBLY.

STANFORD ARTIFICIAL INTELLIGENCE PROJECT
MEMO AIM-163
STAN-CS-266-72

RECURSIVE DEFINITIONS OF PARTIAL FUNCTIONS AND THEIR COMPUTATIONS

BY
J. M. CADIOU

SUPPORTED BY
INSTITUT DE RECHERCHE D'INFORMATIQUE
ET D'AUTOMATIQUE
AND
ADVANCED RESEARCH PROJECTS AGENCY
ARPA ORDER NO. 457

MARCH 1972

COMPUTER SCIENCE DEPARTMENT

School of Humanities and Sciences
STANFORD UNIVERSITY

STANFORD ARTIFICIAL INTELLIGENCE PROJECT
MEMO AIM-163

MARCH 1972

COMPUTER SCIENCE DEPARTMENT
REPORT CS-266

RECURSIVE DEFINITIONS OF PARTIAL FUNCTIONS AND THEIR COMPUTATIONS

by

J.M. Cadiou

ABSTRACT: A formal syntactic and semantic model is presented for 'recursive definitions' which are generalizations of those found in LISP for example. Such recursive definitions can have two classes of fixpoints, the strong fixpoints and the weak fixpoints, and also possess a class of computed partial functions.

Relations between these classes are presented: fixpoints are shown to be extensions of computed functions. More precisely, strong fixpoints are shown to be extensions of computed functions when the computations may involve "call by name" substitutions; weak fixpoints are shown to be extensions of computed functions when the computation only involves "call by value" substitutions. The Church-Rosser property for recursive definitions with fixpoints also follows from these results.

Then conditions are given on the recursive definitions to ensure that they possess least fixpoints (of both classes), and computation rules are given for computing these two fixpoints: the 'full' computation rule, which leads to the least strong fixpoint, and the 'standard innermost' computation rule, which leads to the least weak fixpoint. A general class of computation rules, called 'safe innermost', also lead to the latter fixpoint. The "leftmost innermost" rule is a special case of those, for the LISP recursive definitions.

This research was supported in part by the Institut de Recherche d' Informatique et d'Automatique and in part by the Advanced Research Projects Agency.

The views and conclusions contained in this document are those of the author and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the Advanced Research Projects Agency.

Reproduced in the USA. Available from the National Technical Information Service, Springfield, Virginia 22151.

ACKNOWLEDGEMENT

I would especially like to thank my adviser, Professor Z. Manna for his invaluable help during this research. His constant support and generous contribution of time and effort were beyond the simple call of duty and are deeply acknowledged.

Special thanks also go to Professor R. W. Floyd for his penetrating remarks on an early version of this work which led to some rewarding developments. I am also indebted to Robin Milner and Jean Vuillemin for many stimulating discussions and constructive criticisms.

I am most grateful to Mrs. Queenette Baur and Mrs. Lucy Bertaccini, of the Stanford A.I. Project, for their excellent typing of this dissertation.

I also wish to express my thanks to the Institut de Recherche d'Informatique et d'Automatique and to the Stanford A.I. Project for their support of this work.

Finally, I dedicate this Thesis to my wife Natacha.

TABLE OF CONTENTS

		PAGE
ACKNOWLEDGEMENT		iii
TABLE OF CONTENTS		iv
ADVICE TO THE READER		vi
CHAPTER 1:	GENERAL PRESENTATION AND DISCUSSION OF THE RESULTS	1
1.1	Introduction	2
1.2	Motivation	4
1.3	Terms and their Evaluations	8
1.4	Recursive Definitions	11
1.5	Relations between the Fixpoints and the Computed Functions	15
1.6	Monotonically Structured Recursive Definitions	18
1.7	Fixpoint Computations	20
1.8	Pitfall Computations	25
CHAPTER 2:	THE MODEL : FORMAL DEFINITIONS	29
2.1	Introduction	30
2.2	Syntax	30
2.3	Semantics	33
2.4	Computations	43
CHAPTER 3:	RELATIONS BETWEEN COMPUTED FUNCTIONS AND FIXPOINTS	49
3.1	Introduction	49

PAGE

3.2 Computed Functions and Strong Fixpoints 49

3.3 Innermost Computed Functions and Weak Fixpoints..... 56

CHAPTER 4: MONOTONICALLY STRUCTURED RECURSIVE DEFINITIONS..... 60

4.1 Introduction 61

4.2 Monotonicity Continuity 61

4.3 Monotonically Structured Terms 67

4.4 Existence and Characterization of Least Fixpoints 76

CHAPTER 5: FIXPOINT COMPUTATIONS 78

5.1 Introduction 79

5.2 Standard Simplifications 79

5.3 Full Computations 87

5.4 Standard Innermost Computations 93

5.5 Safe Innermost Computations 103

CONCLUSION 120

APPENDIX I: MONOTONICITY, CONTINUITY. APPLICATION TO PARTIAL FUNCTIONS 122

APPENDIX II: SYSTEMS OF RECURSIVE DEFINITIONS 142

BIBLIOGRAPHY 155

ADVICE TO THE READER

The essence of the material in this work is presented and discussed in Chapter 1. Chapters 2 - 5 contain detailed and formal definitions, together with all the proofs of the results. They are primarily intended to be reference material. The conclusion contains some remarks on possible further research related to this work.

CHAPTER 1

GENERAL PRESENTATION AND DISCUSSION OF THE RESULTS

1.1 Introduction

1.2 Motivation

1.3 Terms and Their Evaluations

- 1.3.1 Syntax
- 1.3.2 Semantics
- 1.3.3 Evaluations of Terms

1.4 Recursive Definitions

- 1.4.1 Fixpoints of Recursive Definitions
- 1.4.2 Computations with Recursive Definitions

1.5 Relations Between the Fixpoints and the Computed Functions

1.6 Monotonically Structured Recursive Definitions

1.7 Fixpoint Computations

- 1.7.1 Full Computation
- 1.7.2 Standard Computation
- 1.7.3 Safe Innermost Computations

1.8 Pitfall Computations

1.1 Introduction

This dissertation presents a syntactic and semantic model for 'recursive definitions' which are generalizations of those introduced in McCarthy (1963).*/

Each 'recursive definition' yields two classes of fixpoints**/, the strong fixpoints and the weak fixpoints, and a class of computed partial functions obtained by applying different computation rules to the 'recursive definition'. In this work we first show the relations between the computed functions and the strong fixpoints (Theorem 1), and between the functions computed by "innermost substitutions" and the weak fixpoints (Theorem 2).

We are, of course, interested in those fixpoints which can be computed. We give a sufficient condition on 'recursive definitions' to guarantee the existence of a partial function which is the least (defined) strong fixpoint of the 'recursive definition' (Theorem 3), and also the existence of a partial function which is the least weak fixpoint of the 'recursive definition' (Theorem 4). We then describe a computation rule for computing the least strong fixpoint (Theorem 5) and another for computing the least weak fixpoint (Theorem 6). We finally give an additional class of computation rules which lead to the least weak fixpoint (Theorem 7).

---

*/ We use quotation marks to avoid possible confusion with the well established meaning of the word "recursive" in computability theory. Our 'recursive definitions' include as special cases those used by Kleene (1936) and others to define recursive functions, but some of our results are actually more general. We will omit the quotation marks when no confusion can arise.

**/ Informally speaking, a fixpoint of a recursive definition is any partial function which satisfies the definition.

Recursive definitions were introduced long ago, for example by Herbrand (1931), Gödel (1934), Kleene (1936) and, in a different way, by McCarthy (1963). More recently, recursive definitions have been studied by several authors:

(a) Morris (1968) emphasizes the distinction between the two ways in which a recursive definition can be considered, namely as defining fixpoint functions or as a means of defining computations.

He also mentions the two possible types of fixpoints, and states a theorem and a conjecture, both of which can be shown as special cases of our results.

(b) Manna and McCarthy (1970) present two computation rules which they call "sequential" and "parallel", and give an example to show that there are other intuitively appealing computation rules. The "sequential" and "parallel" rules, as well as the one suggested by their example are special cases of the rules that we present in Section 1.4.

(c) Manna and Pnueli (1970) formalize computations with recursive definitions as sequences of terms. Our formal computations are generalizations of theirs.

(d) Rosen (1971) establishes, in particular, the Church-Rosser property for recursive definitions which are essentially equivalent to those that we consider in Section 1.6. This property is also a consequence of our results of Sections 1.5 and 1.6. Rosen further establishes that such recursive definitions possess what we call a strong fixpoint (which essentially solves the conjecture in Morris (1968)). Our proof of this result is quite different, and has the advantage of being constructive, that is of exhibiting a computation rule which actually yields the fixpoint in question.

1. Motivation

This section presents some elementary examples and attempts to provide an intuitive background for our model of 'recursive definitions'.

Let us consider first the well-known recursive definition of the factorial function over the integers:

$$F(x) \Leftarrow (\text{if } x = 0 \text{ then } 1 \text{ else } x \cdot F(x-1)). \qquad (1)$$

As emphasized in Morris (1968), and Rosen (1971), there are two different ways in which one can consider this definition:

(a) as a functional equation in the variable "F", (the equation being implicitly universally quantified over "x");

(b) as a recursive definition of F whereby $F(x)$ is defined in terms of $F(x-1)$.

It is essential to understand the difference between the two approaches.

(a) In the first approach, one is interested in studying the partial functions "F" that satisfy the corresponding equation, i.e., the fixpoints of the equation. f is a fixpoint of the equation if:

$$\forall n \,.\, f(n) = (\text{if } n = 0 \text{ then } 1 \text{ else } n \cdot f(n-1)) \qquad (1)'$$

For this equation to be meaningful, one must define precisely the value of the right hand side of (1)' for a given partial function f and for a given value n of x. Let us temporarily denote this value $\tau(f,n)$.

If we take:

$$f(n) = \begin{cases} n! & \text{if } n \geq 0 \text{ } \underline{*}/ \\ 0 & \text{if } n < 0, \end{cases}$$

we see that there is no difficulty in defining $\tau(f,n) = (\text{if } n = 0 \text{ then } 1 \text{ else } n \cdot f(n-1))$, if we consider "if...then...else..." as a function of three

---

$\underline{*}/$ Where n! is defined, for example, as being the number of permutations of n objects, or by some infinite table from which the values are read.

arguments in the obvious way (conditional connective). It is easy to see that, for every integer n, $f(n) = \tau(f,n)$, i.e., f is a fixpoint of (1).

Since partial functions can take the undefined value ω [*], we must be able to give values to expressions which contain undefined arguments. For example, if we take:

$$g(n) = \begin{cases} n! & \text{if} \quad n \geq 0 \\ \omega & \text{if} \quad n < 0 \end{cases}$$

we have for n = 0:

$$\tau(g,0) = (\text{if T then 1 else } 0*\omega).$$

If we define $0*\omega = \omega$, we obtain:

$$\tau(g,0) = (\text{if T then 1 else } \omega).$$

Now, if we define in addition (if T then 1 else ω) = 1, we finally get:

$$\tau(g,0) = 1.$$

In fact, the standard definition of the operation '*' and 'if-then'else' can be extended for ω in such a way that g is also a fixpoint of equation (1).

In general, even after fixing the meaning of the base functions for all possible values of the arguments (including ω), a 'recursive definition' may have more than one fixpoint (possibly infinitely many) -- or none at all.

(b) In the second approach, one is interested in defining ways of actually computing F by using the recursive definition. For example, let us describe the natural way to compute F(2) by using the recursive definition (1):

$F(2) \rightarrow$ if $2 = 0$ then 1 else $2*F(1)$

$\rightarrow 2*F(1)$ (since $2 = 0$ is false)

---

[*] The undefined value ω can be considered as the value associated with a non-terminating computation.

$\rightarrow 2*(\underline{\text{if}}\ 1 = 0\ \underline{\text{then}}\ 1\ \underline{\text{else}}\ 1*F(0))$

$\rightarrow 2*1*F(0)$ (since $1 = 0$ is false)

$\rightarrow 2*(\underline{\text{if}}\ 0 = 0\ \underline{\text{then}}\ 1\ \underline{\text{else}}\ 0*F(-1))$

$\rightarrow 2*1$ (since $0 = 0$ is true)

$\rightarrow 2.$

Such a computation appears as an alternating succession of two kinds of steps: "Simplification" and "Substitution".

In the simplification setps, all the expressions that do not contain the function variable F are evaluated, expressions of the form "(if T then A else B)" are replaced by "A", and expressions of the form "(if F then A else B)" are replaced by "B".

In a substitution step, an expression of the form "F(n)" is replaced by "(if n = 0 then 1 else n*F(n-1))". A substitution step occurs only when no more simplifications can be performed. (In more complicated situations, substitution steps could involve the replacement of $F(\alpha)$ by the corresponding righthand side of the recursive definition even when α is an expression containing F. This corresponds to a "call by name" computation and will be discussed later in detail).

If such a computation of $F(x)$ terminates with an integer n, we say that the corresponding computed function f takes the value n at x, i.e., $f(x) = n$. If it does not terminate, we say that f is undefined at x, i.e., $f(x) = \omega$.

One should be aware that sometimes by adding simplification rules one may extend the computed function, (i.e., one may change nonterminating computations into terminating ones). For example, the recursive definition:

$$F(x) \Leftarrow (\underline{\text{if}}\ F(x) = 0\ \underline{\text{then}}\ 1\ \underline{\text{else}}\ 1) \qquad (2)$$

over the integers, may yield two different values for $f(0)$, which depend upon whether or not we apply the simplification rule;

'(**if** A **then** B **else** B)' replaced by 'B'.

if we do, we get that $f(0) = 1$, otherwise that $f(0) = \omega$.

The sequence in which the substitutions are performed can also affect the convergence of the computation. Consider for example the following recursive definition (Morris (1968), p46):

$$F(x,y) \Leftarrow \underline{\text{if}}\ x = 0\ \underline{\text{then}}\ 0\ \underline{\text{else}}\ 1 + F(x-1,F(y-2,x)) \qquad (3)$$

Let us compute $F(2,1)$. The first step is:

$$F(2,1) \rightarrow 1 + F(1,F(-1,2))$$

Now we see that there are two ways of continuing the computation, according to which occurrence of F we choose to replace by the recursive definition:

(i) if we choose the "innermost" one, the next term is:

$$1 + F(1,1+F(-2,F(0,-1)))$$

(ii) if we choose the "outermost" one, the next term is:

$$2 + F(0,F(F(-1,-2),1))$$

In both cases we have three ways to continue.

It is easy to see that if one keeps substituting for the outermost occurrence of F, the computed value is $f(2,1) = 2$, whereas if one keeps substituting for the innermost occurrence of F, the computation does not terminate, i.e., $f(2,1) = \omega$, (which incidentally, shows that the innermost

computations are not, in general, those that terminate whenever possible).

So in general, different computation mechanisms can lead to different results, and therefore we have a class of partial functions computed with a 'recursive definition'. Our aim in this work is to study the relation of these computed functions with the fixpoints of the 'recursive definitions'.

We will at this point introduce some of the formalism of our system.

1.3 Terms and Their Evaluations

1.3.1 Syntax

The terms which are used in our 'recursive definitions' and in the formal computations are constructed from the symbols of an alphabet consisting of a unary function variable F, an individual variable x, [*] given n-ary function letters g_i, individual constants c_i, and ω (which stands for the symbol of an "undefined" element), as follows:

(a) x, c_i and ω are terms.

(b) if $\alpha_1, \alpha_2, \ldots, \alpha_n$ are terms, and g_i is an n-ary given function letter, then $g_i(\alpha_1, \alpha_2, \ldots, \alpha_n)$ is a term.

(c) if α is a term, then $F(\alpha)$ is a term.

(d) there are no other terms than those which can be obtained from (a) by a finite number of applications of (b) and (c).

For example, $g(F(x), g(c, \omega))$ is a term.

1.3.2 Semantics

We define the semantics of the system by interpreting, in the natural way, the c_i's as elements c_i of some domain Δ, ω as the undefined element ω [**] and the g_i's as partial functions g_i, as explained on the following page.

---

[*] For convenience, we only discuss in this Chapter recursive definitions with a single function variable and a single individual variable. In Chapters 2, 3, 4 and 5 the results are stated and proved for recursive definitions with several individual variables. The results can be extended to systems of such recursive definitions (See Appendix II).

[**] ω has in fact the formal properties of an expression whose computation does not terminate.

A subset D of Δ ($D \subseteq \Delta$) is also specified as part of the interpretation. It is the domain of the unknown partial functions f's.

Note that an element of the alphabet is represented by a letter underlined with a '$\underline{\ }$', whereas its interpretation is represented by the same letter without underline. We will omit the underline '$\underline{\ }$' whenever no confusion can arise.

The interpretation of x and F will be discussed in the next paragraph.

There is a technical difficulty involved in the interpretation of the $\underline{g}_i$'s : in most cases, one is interested in computing partial functions over some domain D (e.g., the integers, the lists, etc.), but one needs to evaluate some terms outside of that domain (for example, predicates). To cope with this problem, Manna and Pnueli (1970) define two kinds of terms in their syntax: the conditional terms (which evaluate in D^+ [*]) and the propositional terms (which evaluate in $\{T,F\}^+$). Instead, we let the domain D of the partial functions be a subdomain of Δ, the domain of our semantic interpretation. In most of our examples, Δ is $D \cup \{T,F\}$. In general, each n-ary given function is specified as a partial function over a subdomain of $(\Delta^+)^n$. Then, as shown below, the conditional connectives can be introduced as given functions, and so we do not make a special syntactic treatment for them.

Example: The conditional connective 'if-then-else' is usually understood as a ternary function over $\{T,F\}^+ \times (D^+)^2$, which is a subdomain of $(\Delta^+)^3 = (D \cup \{T,F\} \cup \{\omega\})^3$, and can be defined in several ways.[**] Two variants of the 'if-then-else' connective have been discussed by Manna and McCarthy (1970):

(a) the sequential 'if-then-else', where, for every $x,y \in D^+$: [***]

$$(\text{if } T \text{ then } x \text{ else } y) \equiv x$$

---

[*] D^+ stands for $D \cup \{\omega\}$; in general, for any set S, we denote $S \cup \{\omega\}$ by S^+.

[**] Notice that we use only one symbol ω, instead of using one for D and one for the truth value domain $\{T,F\}$. In fact, the interpretation is not constrained to have a truth value domain $\{T,F\}$ distinct from D. The formal treatment of chapters 2-5 shows that this does not weaken the results.

[***] $\equiv$ indicates the equality relation over Δ^+; therefore $\omega \equiv \omega$ is true.

$(\text{if } F \text{ then } x \text{ else } y) = y$

$(\text{if } \perp \text{ then } x \text{ else } y) = \perp$

(b) the parallel 'if-then-else', where, for every $x,y \in D^+$:

$(\text{if } T \text{ then } x \text{ else } y) = x$

$(\text{if } F \text{ then } x \text{ else } y) = y$

$(\text{if } \perp \text{ then } x \text{ else } x) = x$, and

$(\text{if } \perp \text{ then } x \text{ else } y) = \perp$ when $x \neq y$.

Note that the parallel 'if-then-else' is more defined than the sequential 'if-then-else'.

1.3.3 Evaluations of Terms

Informally speaking, once the variables F and x are specified (i.e., bound to given values), a term $\tau(F,x)$ can be evaluated in the obvious way (i.e., the inner subterms are evaluated first).

There are two natural ways of specifiying x and F:

(a) one is to specify x as an element of D and F as a partial function over D (i.e., a mapping of D into D^+).

(b) the other is to specify x as an element of D^+ [*] and F as a partial function over D^+ (i.e., a mapping of D^+ into D^+). [**]

Corresponding to these specifications of the variables, there are two ways of evaluating the terms, which yield what we call, respectively, the weak value and the strong value of a term.

(a) The weak value of $\tau = \tau(F,x)$ for F being $f \in pf(D)$ [***] and x being $\xi \in D$ is an element of Δ^+, denoted $\tilde{\tau}(f,\xi)$, which is defined inductively as follows:

---

[*] One may want to allow x to range over D^+ (i.e., with possibly undefined value), because one may want to consider x as the value of a computation (which may be nonterminating).

[**] What we call a "partial function over D^+" is actually what is usually called a total function over D^+ (e.g., in Morris (1968)). However, we find it convenient to consider such a function as a special case of a partial function of a domain S into a range R, where $S = D^+$ and $R = D$. See Appendix I for a presentation of the main results regarding those functions.

[***] We denote the set of partial functions over D by $pf(D)$ and the set of partial functions over D^+ by $pf(D^+)$.

(i) if τ is x, $\tilde{\tau}(f,\xi) = \xi$,

(ii) if τ is c_i, $\tilde{\tau}(f,\xi) = c_i$,

(iii) if τ is α, $\tilde{\tau}(f,\xi) = \alpha$,

(iv) if τ is $g_i(\tau_1,\ldots,\tau_n)$, $\tilde{\tau}(f,\xi) = g_i(\tilde{\tau}_1(f,\xi),\ldots,\tilde{\tau}_n(f,\xi))$, */

(v) if τ is $F(\alpha)$ there are two cases:

(v.1) if $\tilde{\tau}(f,\xi) = \alpha$ then $\tilde{\tau}(f,\xi) = \alpha$ (this special rule and its effects are discussed below);

(v.2) if $\tilde{\tau}(f,\xi) \neq \alpha$ then $\tilde{\tau}(f,\xi) = f(\tilde{\tau}(f,\xi))$.

(b) the <u>strong value</u> of $\tau = \tau(F,x)$ for F being $f \in pf(D^+)$ and x being $\xi \in D^+$ is an element of Δ^+, denoted $\hat{\tau}(f,\xi)$, which is defined inductively as follows:

(i) - (iv): as in case (a)

(v) if τ is $F(\alpha)$ then $\hat{\tau}(f,\xi) \equiv f(\hat{\alpha}(f,\xi))$

The only difference between the two evaluations resides in the evaluation of the terms $F(\alpha)$. In the first case, since f is a mapping of D into D^+, $f(\alpha)$ has no meaning: the solution adopted is equivalent to arbitrarily setting $f(\alpha) = \alpha$. In the second case, since f is a mapping of D^+ into D^+, $f(\alpha)$ is specified.

This difference turns out to be an essential one, as will be seen in Section 1.4.1.

1.4 <u>'Recursive Definitions'</u>

A 'recursive definition' is an expression of the form $F(x) = \tau(F,x)$, where τ is a term over an interpreted alphabet. We use D to denote the partial function domain of the interpretation.

1.4.1 <u>Fixpoints of 'Recursive Definitions'</u>

Let $F(x) = \tau(F,x)$ be a 'recursive definition'. There are two types

*/We are implicitly assuming here that the vector $\langle \tilde{\tau}_1(f,\xi),\ldots,\tilde{\tau}_n(f,\xi) \rangle$ belongs to the domain of g_i. In Chapter 2, we present a method for relaxing this assumption.

of fixpoint functions of this 'recursive definition', corresponding to the two evaluations of $\tau(F,x)$ discussed in the previous paragraph:

(a) A partial function f over D is a <u>weak fixpoint</u> of the 'recursive definition' if:

$$\forall \underline{\xi} \in D : f(\underline{\xi}) \simeq \tilde{\tau}(f,\underline{\xi}) ;$$

(b) A partial function f over D^+ is a <u>strong fixpoint</u> of the 'recursive definition' if:

$$\forall \underline{\xi} \in D^+ : f(\underline{\xi}) \simeq \tilde{\tau}(f,\underline{\xi}).$$

These two types of fixpoints are in general completely different: there are 'recursive definitions' [*] which have weak fixpoints but no strong fixpoints; others have strong fixpoints but no weak fixpoints. There are even recursive definitions which have weak fixpoints and strong fixpoints which agree nowhere on D. Examples of such recursive definitions can be found in Section 2.5. of Chapter 2. However, we shall see (Section 1.7.2) that, for an important subclass of 'recursive definitions', the weak fixpoints and the strong fixpoints of such recursive definitions have the property that they are all extensions [**] of one of them, the least weak fixpoint of the recursive definition. A weak (resp. strong) fixpoint f of a given 'recursive definition' is said to be a <u>least</u> weak fixpoint (resp. strong) of the 'recursive definition' if every weak (resp. strong) fixpoint of the recursive definition is an extension [**] of f.

1.3. <u>Computations with Recursive Definitions</u>

A natural way of formalizing computations such as the ones we informally presented in Section 1.1 is to consider them as a sequence of

---

[*] We emphasize again that we use the word "recursive" to indicate that the function variable appears in both sides of the "recursive definition", i.e. that F is defined in terms of itself, which is the usual intended meaning in programming languages (e.g. recursive procedure). It is not intended to mean that $\tau(F,x)$ is recursive in F or x in the sense of the recursive function theory. In fact, so far, the base functions of the model may well be non-computable. Restrictions on the base functions are only needed for Theorems 2-7.

[**] If f and g are partial functions over a set S (i.e mappings of S into S^+), we say that g is an <u>extension</u> of f iff: $(\forall x \in S)\,[f(x) \neq \omega \Rightarrow f(x) = g(x)]$. See Appendix I for general properties of this extension relation.

terms, transformed by successive manipulations, in the style of Manna and Vuillemin (1970).

Given a 'recursive definition'

$$F(x) \Leftarrow \tau(F,x),$$

we define a computation of a term $\alpha(F,x)$ for $x = c$, where c is an element of D^+, as a sequence of terms α_i, where:

$$\alpha_0 = \alpha(F,\underline{c}),$$

and, for $i \geq 0$, α_{i+1} is deduced from α_i by a finite number of applications of two basic rules:

1. Replacement of a term β occurring in α_i by some term γ where, for every partial function f over D^+

$$\tilde{\beta}(f,c) = \tilde{\gamma}(f,c),$$

i.e., replacements that must preserve equality of the strong values. */

2. Replacement of a term of the form $F(\sigma)$ occurring in α_i by (F,σ) where σ can be any term.

A computation terminates if and only if it eventually reaches some value in Δ. (If the computation reaches ω it is not considered to have terminated.)

Rule 1 allows most familiar simplification rules, such as:

(a) for the sequential 'if-then-else' connective:

$$(\underline{\text{if}}\ T\ \underline{\text{then}}\ A\ \underline{\text{else}}\ B) \rightarrow A,$$

$$(\underline{\text{if}}\ F\ \underline{\text{then}}\ A\ \underline{\text{else}}\ B) \rightarrow B, \text{ and}$$

$$(\underline{\text{if}}\ \omega\ \underline{\text{then}}\ A\ \underline{\text{else}}\ B) \rightarrow \omega\ ;$$

(b) for the parallel 'if-then-else' connective:

$$(\underline{\text{if}}\ A\ \underline{\text{then}}\ B\ \underline{\text{else}}\ B) \rightarrow B\ ;$$

(c) for the multiplication function '*' over the integers extended by $\omega * 0 = 0$:

*/ Notice that the definition of a computation demands that an interpretation of the alphabet be supplied.

$\underline{0} * A \rightarrow \underline{0}$, and

$A * \underline{0} \rightarrow \underline{0}$;

(d) for the multiplication function '$*$' over the integers extended by $1 * \omega = \omega * 1 = \omega$

$\underline{1} * A \rightarrow A$, and

$A * \underline{1} \rightarrow A$.

It also allows the replacement of given functions (with completely specified arguments) by their values, that is:

$$g(\underline{a}_1, \ldots, \underline{a}_n) \rightarrow \underline{b} \quad \text{where } b = g(a_1, \ldots, a_n).$$

But in fact there are many 'non-standard' replacements that are allowed by Rule 1. For example,

(a) "Backward" calculations, such as:

$A \rightarrow$ (if T then A else B), or

(b) Replacements indicating more sophisticated "logical deductions", such as:

(if $F(x) = 0$ then $F(x)$ else 0) $\rightarrow 0$.

This last example is the one that Manna and McCarthy (1970) use to suggest appealing computation rules other than their 'sequential' and 'parallel' rules.

It should be emphasized that, because of Rule 1, computations depend on the interpretation of the alphabet: some replacements which are allowed within one interpretation can become illegal in another interpretation.

Rule 2 clearly allows not only the "innermost" substitutions of $F(\alpha)$ by $\tau(F,\alpha)$ (that is, when α is a constant), but also any "outer" substitution of a term $F(\alpha)$ by $\tau(F,\alpha)$ (that is, when α is an arbitrary term which may contain F's). Morris (1968) and Rosen (1971) use the phrase "call by name" to designate these "outer" substitutions.

Intermediate steps can be skipped in a computation. For example, one may replace an occurrence of $F(x)$ by its computed value, assuming it is known from previous computations.

For a given 'recursive definition', there are of course many ways in which one could apply these rules to a given term, and consequently there may be a large number of different computations of this term.

So in general, there are many computed functions of a 'recursive definition'. A partial function f over D^+ (or over D) is said to be a <u>computed function</u> of a given recursive definition over D^+ (or over D) if it is such that, for every $x \in D^+$ (or for every $x \in D$):

(i) if $f(x) \neq \perp$ then there is a computation of $F(x)$ which terminates with $f(x)$,

(ii) if $f(x) = \perp$ then there is a computation of $F(x)$ which does not terminate. *

1.5 <u>Relations Between the Fixpoints and the Computed Functions</u>

We now come to two important results, relating the fixpoints of a given 'recursive definition' to its computed functions.

<u>THEOREM 1</u>. <u>For every 'recursive definition', every strong fixpoint is an extension of every computed function over D^+</u>.

<u>THEOREM 2</u>. <u>For every 'recursive definition', every weak fixpoint is an extension of every computed function over D obtained by innermost computation</u>. **

---

*/ It is clear that "computed functions" as we have defined them may not be "computable" in the usual computability theory meaning. This makes theorems 1 and 2 stronger. Notice also that the notion of "computed function" is relative to a particular recursive definition.

**/ An <u>innermost computation</u> is any computation in which only <u>call by value</u> is permitted, that is, $F(\alpha)$ may be replaced by $\tau(F,\alpha)$ only when α is an individual constant.

The proofs of these results are given in Chapter 5.

Theorem 1 implies that:

(a) If one of the computed functions is a strong fixpoint, then it is a least strong fixpoint.

Therefore, if a 'recursive definition' does not have a least strong fixpoint, no computed function can be a strong fixpoint of this 'recursive definition'.

(b) A 'recursive definition' cannot have more than one 'computable' strong fixpoint (that is, a computed function which is a strong fixpoint).

(c) If a 'recursive definition' has a strong fixpoint, and if two computations for a given x in D^+ terminate, they must terminate with the same value. This is a Church-Rosser type result, and it also appears in Rosen (1971). Rosen's proof of this corollary is entirely different from ours, and does not rely on the fixpoint idea.

(d) The result of any terminating computation of $F(x)$ must be the value at x of every one of the strong fixpoints of the 'recursive definition'. This implies that:

(1) If a 'recursive definition' has a strong fixpoint which is undefined on some subdomain D' of D^+, then no computation of $F(x)$ can terminate on D'.

Example: Consider again the recursive definition of the factorial function over the integers:

$$F(x) \Leftarrow (\underline{\text{if}}\ x = \underline{0}\ \underline{\text{then}}\ \underline{1}\ \underline{\text{else}}\ x \cdot F(x-\underline{1})).$$

In section 1.2 we mentioned that the partial function:

$$g(n) = \begin{cases} n! & \text{if } n \geq 0 \\ \perp & \text{if } n < 0 \end{cases}$$

was a fixpoint of the recursive definition for the appropriate interpretation of the given functions. It follows therefore that no computation of $F(x)$ using this recursive definition can terminate for $x < 0$.

(2) In particular, it follows that no 'recursive definition' for which the totally undefined function Ω is a strong fixpoint can have terminating computations of $F(x)$, for any x.

Thus, if F occurs in τ, and if all the given functions g used in $\tau(F,x)$ are such that $g(\ldots,\omega,\ldots) = \omega$ (i.e., are undefined whenever at least one argument is undefined), then it can be shown that Ω is a strong fixpoint of $F(x) \Leftarrow \tau(F,x)$, and therefore no computation of the 'recursive definition' can terminate.

For example, for $F(x) \Leftarrow F(x)$, $F(x) \Leftarrow F(x+1)$ or $F(x) \Leftarrow F(x) + 1$, (where $\omega + 1 = \omega$), no computation can terminate.

This implies that in a 'recursive definition' having a terminating computation at least one given function has to be sometimes defined when one of its arguments is undefined. Indeed, both the sequential and parallel 'if-then-else' connective have this property.

(3) If a 'recursive definition' has two strong fixpoints that differ for some x_0, then no computation of $F(x_0)$ can terminate.

Example: Consider the following 'recursive definition' over the natural numbers:

$$F(x) \Leftarrow \underline{\text{if}}\ (F(x) = F(x \dot{-} \underline{1}) + \underline{1})\ \underline{\text{then}}\ x\ \underline{\text{else}}\ \underline{0}$$ */

with the standard interpretation of $=$, $\dot{-}$ (with $0 \dot{-} 1 \equiv 0$ and $\omega \dot{-} 1 \equiv \omega$), $+$ (with $\omega + 1 = \omega$), and the parallel 'if-then-else' connective. It can be

*/ Because of the presence of $=$, the righthand side of this definition is not "recursive" in the sense of the usual computability theory.

shown that both the partial functions $f(n) = 0$ and $g(n) = n$ are strong fixpoints of this 'recursive definition'. Therefore, no computation of $F(x)$ can terminate for $x > 0$.

Similar consequences can be obtained from Theorem 2 for the weak fixpoints and the functions computed by innermost computations.

By combining Theorem 1 and Theorem 2, one gets for example that, if a 'recursive definition' has a weak fixpoint and a strong fixpoint which do not agree for some $x_0 \in D$, then no innermost computation of $F(x)$ can terminate for x_0.

It is very easy to give examples where a computed function is not a strong fixpoint of the 'recursive definition', or where a function computed by innermost computation is not a weak fixpoint of the 'recursive definition'. We now want to give sufficient conditions for fixpoint computations (i.e., computations such that the computed function is a fixpoint of the 'recursive definition').

Since there are 'recursive definitions' without fixpoints, these conditions must obviously concern both the 'recursive definitions' and the computation rules.

1.6 Monotonically Structured Recursive Definitions

As indicated above, there are 'recursive definitions' which have no fixpoints at all, such as:

$$F(x) \Leftarrow \underline{\text{if}}\ F(x) = \underline{0}\ \underline{\text{then}}\ \underline{1}\ \underline{\text{else}}\ \underline{0}\ .$$

There are also 'recursive definitions' which have strong fixpoints for example, but do not have any that are computed functions over D^+. For example, the 'recursive definition':

$$F(x) \Leftarrow \underline{\text{if}}\ (F(x) = F(x \dot{-} \underline{1}) + \underline{1})\ \underline{\text{then}}\ x\ \underline{\text{else}}\ \underline{0}\ ,$$

(where the given functions have the same interpretation as in the last example of section 1.5), has strong fixpoints,

e.g., $f(n) \equiv n$ [*], but does not have any least fixpoint, and therefore none of these fixpoints can be a computed function of the 'recursive definition'.

We will now give a <u>sufficient condition</u> on 'recursive definitions' which guarantees the existence of a <u>least strong fixpoint</u> and of a <u>least weak fixpoint</u>.

We first construct an ordering $\prec$ on $(\Delta^+)^n$ in the following way (see Scott (1969)):

(a) $\omega \prec x$ and $x \prec x$ for every $x \in \Delta^+$ (all other pairs are unrelated)

(b) $(x_1, x_2, \ldots, x_n) \prec (y_1, y_2, \ldots, y_n)$ iff

$(x_1 \prec y_1)$ and $(x_2 \prec y_2)$ and ... and

$(x_n \prec y_n)$.

We say that <u>a term α is a monotonically structured</u> when every given function occurring in α is a monotonic in the sense that is preserves the ordering $\prec$ defined above.

Base functions, g's such that $g(\ldots,\omega,\ldots) = \omega$ (i.e., that are undefined when at least one of their arguments is undefined, regardless of the values of the other arguments) are monotonic. The sequential and parallel if-then-else connectives are also monotonic; but $=$ (the equality predicate over Δ^+) is <u>not</u> monotonic.

Any function g whose values are known for defined arguments can be extended on undefined arguments in such a way that it becomes monotonic, (e.g. by defining $g(\ldots, \omega, \ldots) = \omega$ as above. [**]

---

[*] Actually, one can show that the strong fixpoints of this recursive definition are all the functions:

$$f_i(x) = \begin{cases} \omega \text{ if } x = \omega \\ x \text{ if } 0 \leq x \leq i \\ 0 \text{ if } x > i \end{cases} \quad \text{and} \quad g_i(x) = \begin{cases} 0 \text{ if } x = \omega \\ x \text{ if } 0 \leq x \leq i \\ 0 \text{ if } x > i \end{cases}$$

for all i, $0 \leq i \leq \infty$.

[**] In fact, one can show that if g is a partial function over Δ^p, it has a minimal monotonic extension to (Δ^+), which is the one discussed above, and a maximal monotonic extension to $(\Delta^+)^p$. In general it is not possible to obtain the maximal extension in an effective way.

We say that a recursive definition $f(x) \Leftarrow \tau(F,x)$ is monotonically structured if $\tau(F,x)$ is monotonically structured. This class of recursive definitions contains as special cases:

(i) Kleene's definition of partial recursive functions.

(ii) The McCarthy calculus with sequential and parallel 'if-then-else' connectives.

The interest of monotonically structured recursive definitions lies in the following two theorems:

THEOREM 3. A monotonically structured recursive definition has a least strong fixpoint.

THEOREM 4. A monotonically structured recursive definition has a least weak fixpoint.

A characterization of these fixpoints in terms of least upper bounds will be found in Chapter 4.

1.7 Fixpoint Computations

In the remainder of this Chapter, we describe computation rules which lead to the least strong (or weak) fixpoint for monotonically structured recursive definitions, thus showing that the fixpoints of Theorems 3 and 4 are in fact computable.

1.7.1. Full Computation

The full computation of $\alpha(F,x)$ for $x = x_0$ is a sequence of terms α_i, starting with $\alpha_0 = \alpha(F,x_0)$ and such that α_{i+1} is obtained from α_i by:

(a) first performing all possible standard simplications, and then:

(b) simultaneously substituting for all the occurrences of F in α_i. */

A standard simplification is any rule of the form:

$$g(A_1,A_2,\ldots,A_n) \rightarrow a ,$$

*/This is indeed possible, as shown in Section 5.3.

where :

(i) each A_i is an individual constant or a term variable (a term variable is a letter which stands for an arbitrary term);

(ii) no two term variables are identical;

(iii) $\underset{\sim}{a}$ is an individual constant (not $\underset{\sim}{\omega}$);

(iv) the equation $g(\xi_1,\xi_2,\ldots\xi_n) \equiv a$ holds, for every n-tuple $< \xi_1,\ldots\xi_n >$ in the domain of g such that, if A_i is an individual constant $\underset{\sim}{a_i}$, ξ_i takes its value a_i.

All rules that satisfy the above definition must appear in the set of standard simplifications.

Examples: (a) if T then 1 else A $\rightarrow \underset{\sim}{1}$ is a standard simplification since: if T then 1 else $\omega = 1$.

(b) If '*' is the ordinary multiplication over the integers, then $\underset{\sim}{3}*\underset{\sim}{2} \rightarrow \underset{\sim}{6}$ is a standard simplification corresponding to the equation $3*2 = 6$. This is a case where all the A_i's of the definition are individual constants.

(c) If '*' is the ordinary multiplication over the integers extended by $0*\omega \equiv \omega$, then $\underset{\sim}{0}*A \rightarrow \underset{\sim}{0}$ is not a standard simplification, because the equation $0*\omega \equiv 0$ does not hold. However, if '*' is the ordinary multiplication over the integers now extended by $0*\omega \equiv 0$, (which is a monotonic extension) then $\underset{\sim}{0}*A \rightarrow \underset{\sim}{0}$ is a standard simplification.

The rules for standard simplifications are not completely deterministic, since the order in which the simplifications are performed is not specified, but one can show that the final term (after all possible simplifications) is the same, regardless of the order in which the simplifications are performed.

As an example of full computation, let us consider the full computation of the term $\alpha = F(F(\underset{\sim}{0}))$ using the recursive definition of the

factorial over the integers. The sequence of terms of this computation is:

$\alpha_0 = \alpha = F(F(\underset{\sim}{0}))$

$\alpha_1 =$ **if** ((**if** $\underset{\sim}{0} = \underset{\sim}{0}$ **then** $\underset{\sim}{1}$ **else** $\underset{\sim}{0}*F(\underset{\sim}{0}-\underset{\sim}{1})) = \underset{\sim}{0}$) **then** $\underset{\sim}{1}$

else (**if** $\underset{\sim}{0}=\underset{\sim}{0}$ **then** $\underset{\sim}{1}$ **else** $\underset{\sim}{0}*F(\underset{\sim}{0}-\underset{\sim}{1})$)$*F$((**if** $\underset{\sim}{0}=\underset{\sim}{0}$ **then** $\underset{\sim}{1}$ **else** $\underset{\sim}{0}*F(\underset{\sim}{0}-\underset{\sim}{1})$) $-\underset{\sim}{1}$

After standard simplification, α_1 reduces to:

$\alpha_2 = \underset{\sim}{1}*F(\underset{\sim}{0})$, and the following terms are:

$\alpha_3 = \underset{\sim}{1}*$ (**if** $\underset{\sim}{0} = \underset{\sim}{0}$ **then** $\underset{\sim}{1}$ **else** $\underset{\sim}{0}*F(\underset{\sim}{0}-\underset{\sim}{1})$)

$\alpha_4 = \underset{\sim}{1}$.

We have:

THEOREM 5: (full computation). For a monotonically structured recursive definition, the partial function over D^+ computed by full computation is the strong fixpoint of the recursive definition.

1.7.2 Standard Computation

The standard computation of $\alpha(F,x)$ for $x = x_0$ is a sequence of terms α_i starting with $\alpha_0 = \alpha(F,x_0)$, and such that α_{i+1} is obtained from α_i by:

(a) first performing all possible standard simplifications, and then

(b) simultaneously substituting for all the innermost occurrences of F.

THEOREM 6: (standard computation). For a monotonically structured recursive definition, the partial function over D computed by using the standard computation is the least weak fixpoint of the recursive definition.

Notice that, from Theorems 1 and 6, it follows that for a monotonically structured recursive definition, the least strong fixpoint is an extension of the least weak fixpoint. Actually, there are examples for which the least strong fixpoint is strictly more defined then the least weak fixpoint. (Example (3) of Section 1.2 is one such. This example is from Morris (1968)). The full computation uses call by name whereas the standard computation uses call by value; it is usually assumed in software discussion that,

in the absence of side effects, the use of call by name versus the use of call by value affects the rate, but not the existence, of convergence: it follows from the above discussion that this is not true.

1.7.3 Safe Innermost Computations

The standard computation is interesting for theoretical purposes, but it is by no means the only computation rule which yields the least defined weak fixpoint. In fact, it is shown in Chapter 5 that;

THEOREM 7: (safe innermost computation). For a monotonically structured recursive definition, any partial function over D computed using safe innermost computation is the least weak fixpoint of the recursive definition.

A safe innermost computation of $\alpha(F,x)$ for $x = x_0$, is any sequence of terms α_i such that α_0 is $\alpha(F,x_0)$ and α_{i+1} is obtained from α_i by:

(a) first performing a set of safe simplifications (see below).

(b) performing safe innermost substitutions.

A set of safe simplifications consists of the set of standard simplifications (See paragraph 1.7.1) augmented by rules of the form:

$$g(A_1, A_2, \ldots, A_n) \rightarrow B$$

where:

(a) the A_i's are either individual constants or term variables (not necessarily all distinct);

(b) B is an A_i which is a term variable;*/

(c) the equality $g'(x_1,\ldots,x_n) = y$, where g' is a regular**/ extension of g, holds for all possible values of the n+1 -tuple $\langle x_1,\ldots,x_n,y\rangle$ over Δ^+ such that:

(i) $\langle x_1,\ldots,x_n\rangle$ belongs to the domain of g'

*/The case in which B is an individual constant is redundant because one can show that the corresponding rule must already be in the set of standard simplifications.

**/A regular extension of a monotonic given function g is a function g' such that:
1) $\forall \xi \in \mathrm{Dom}(g)$, $g'(\xi) \equiv g(\xi)$
2) $\mathrm{Dom}(g') = \mathrm{Dom}(g) \cup \{\xi \in (\Delta^+)^n \mid \xi < \eta \text{ and } \eta \in \mathrm{Dom}(g)\}$
3) g' is monotonic.
See Chapter 5, paragraph 5.5.4. for details

(ii) For every i, $1 \leq i \leq n$:

if A_i is a constant a_i, then $x_i = \underset{\sim}{a}_i$

if A_i and A_j are the same term variable,

then $x_i = x_j$.

(iii) If B is the term variable A_i, then $y = x_i$.

Note that there is no need that all possible rules satisfying these requirements will be in a particular set of safe simplifications. In other words, one is free to pick none, some or all of these rules to augment the set of standard simplifications.

The safe <u>innermost substitutions</u> are innermost substitutions performed on certain key positions. To define these positions, we first need to define the ω-sets of a base function g; an <u>ω-set</u> of a base function g in a term $\underset{\sim}{g}(\alpha_1,\ldots,\alpha_n)$ is a set of argument positions such that if all are undefined in the set, the corresponding value of g is undefined. More formally, it is any subset I of $\{1,\ldots,n\}$ such that, for every $i \in I$, α_i is not a constant, and the equation: $g(x_1,\ldots,x_n) = \omega$ holds for every n-tuple $\langle x_1,\ldots,x_n \rangle$ in the domain of g such that:

for every $i \in I$, $x_i \equiv \omega$, and:

for every i, $1 \leq i \leq n$, such that α_i is a constant $\underset{\sim}{a}_i$, x_i takes its value a_i.

<u>Example</u>. In the term: if $\underset{\sim}{p}(F,x)$ then $\underset{\sim}{g}(F,x)$ else $\underset{\sim}{h}(F,x)$. the sequential 'if-then-else' connective has five ω-sets: $\{1\}$, $\{1,2\}$, $\{1,3\}$, $\{2,3\}$, $\{1,2,3\}$; the parallel 'if-then-else' connective has only four ω-sets: $\{1,2\}$, $\{1,3\}$, $\{2,3\}$, $\{1,2,3\}$.

Now the key positions in a term α where safe innermost substitutions may be performed, can be determined inductively in the following way:

(a) if $\alpha = x$, $\underset{\sim}{c}$ or ω, there is no substitution to perform;

(b) if $\alpha = \underset{\sim}{g}(\alpha_1,\alpha_2,\ldots,\alpha_n)$, then simultaneously perform the

substitutions in the key positions in α_i for all $i \in I$, where I is any ω-set of g in α (note that this is a non-deterministic process);

(c) if $\alpha = F(\beta)$ we distinguish between two cases:

(i) if β contains F, then perform the substitutions in the key positions in β;

(ii) if β is an individual constant then substitute $\Omega(F,\beta)$ for $F(\beta)$.

The intuitive idea behind this is quite simple: because the ω-sets of a given function $\underset{\sim}{g}$ indicate positions such that when they are undefined the whole function becomes undefined, "in-depth" computing within these positions should be safe -- i.e., computations of these arguments must eventually terminate if the computation of the whole function is to terminate.

The 'sequential' and 'parallel' computation rules of Manna and McCarthy (1970) are safe innermost computations for the recursive definitions that these authors consider.

1.8 Pitfall Computations

In this section, we discuss several examples of 'recursive definitions' and/or computations which do not satisfy the sufficient conditions of Sections 1.6 and 1.7.

Example 1. Consider the recursive definition

$$F(x) \Leftarrow F(x)+1$$

over the integers.

For any integer, the standard computation of $F(n)$ using this definition is:

$$F(n) \rightarrow F(n)+1 \rightarrow F(n)+2 \rightarrow \ldots$$

which never terminates.

It follows that the computed function for this recursive definition is Ω, the totally undefined function. For Ω to be a fixpoint of the recursive

definition, it is however necessary that $\omega = \omega + 1$; if we had $\omega \neq \omega + 1$, then "+" would not be monotonic, which would violate the general requirement of Section 1.6.

Example 2: Let us consider the following recursive definition:

$$F(x) \Leftarrow (\text{if } x = 0 \text{ then } 0 \text{ else } F(x-1) * F(x+1))$$

over the integers.

If '*' is defined so that $0 * \omega = \omega * 0 = \omega$ and 'if-then'else' is the sequential conditional connective, the partial function computed using the standard rule is:

$$\psi(n) = \begin{cases} 0 \text{ if } n = 0 \\ \\ \Omega \text{ otherwise.} \end{cases}$$

$\psi(n)$ is a fixpoint over D of the recursive definition, in accordance with the results of Section 1.7.

Now let us modify '*' so that $0 * \Omega = \Omega * 0 = 0$, and consider several different computation rules:

(a) If we still compute using the same rule as above (i.e., without simplification by 0), then we again obtain $\psi(n)$, which is no longer a weak fixpoint of the equation. In this case, '*' is still monotonic, but we have violated the conditions of our standard computation of Section 1.7. Since the equation:

$$0 * \omega \equiv \omega * 0 \equiv 0$$

holds, we should have used the simplification rules:

$$0 * A \rightarrow 0 \text{ and } A * 0 \rightarrow 0$$

If we use these rules, and if we perform the substitutions in parallel, the computation becomes a standard computation, and we obtain the function:

$$\varphi(n) \equiv 0 \text{ (for all } n)$$

which is the least defined weak fixpoint of the recursive definition.

(b) However, if we use the "leftmost innermost" substitution rule,

with the above simplification rules $0*A \to 0$ and $A*0 \to 0$, we obtain the function:

$$\chi(n) = \begin{cases} 0 & \text{if } n \geq 0 \\ \perp & \text{otherwise} \end{cases}$$

which is not a weak fixpoint of the equation. This method of substitution, however, is not a safe innermost substitution according to Section 1.7, since the only $\perp$ - set of '*' in this case is $\{1,2\}$, which requires that substitutions be performed on both positions.

Example 5: Consider the 'recursive definition':

$$F(x) \Leftarrow (\text{if } F(x) = 0 \text{ then } F(x) \text{ else } 0)$$

where the conditional connective is the parallel 'if-then-else'.

The partial function computed by any of the methods of Section 1.7 will lead to the undefined function. However, Ω is not a fixpoint of the 'recursive definition', since for every x, $\Omega(x) = \perp$, whereas:

$$\tau(\Omega,x) \equiv (\text{if } \perp = 0 \text{ then } \perp \text{ else } 0)$$
$$\equiv (\text{if F then } \perp \text{ else } 0)$$
$$\equiv 0.$$

In this case, $\tau(F,x)$ does not satisfy the requirements of Section 1.6 since τ is not monotonic.

There is one way to get around this -- in this particular case -- which is the following: consider $\tau(F,x)$ to be: $\underline{g}(F(x),\underline{0})$, where $g(y,z) \equiv$ if $y = z$ then y else z). Now it can be checked that g is monotonic and verifies the identity:

$$\forall y,z : g(y,z) \equiv z$$

Therefore, g has the 'safe' simplification rule

$$g(A,B) \to B$$

according to Section 6.

Hence the computation step

$$g(F(x),0) \rightarrow 0$$

is now legal, and leads to the least defined fixpoint over D of the equation, which is the constant function always equal to 0.

CHAPTER 2

THE MODEL: FORMAL DEFINITIONS

2.1 Introduction

2.2 Syntax

- 2.2.1 Alphabet
- 2.2.2 Terms
- 2.2.3 Structural Induction on Terms
- 2.2.4 Substitutions

2.3 Semantics

- 2.3.1 Interpretation of an Alphabet
- 2.3.2 Evaluation of terms
- 2.3.3 Functionals associated with a Term
- 2.3.4 'Recursive Definitions' and their Fixpoints
- 2.3.5 Relations between the two types of evaluations and fixpoints

2.4 Computations

- 2.4.1 Notations
- 2.4.2 Computation of a term with a 'recursive definition'
- 2.4.3 Terminating computations
- 2.4.4 Computed functions
- 2.4.5 Innermost computations

2.1 Introduction

In this chapter, we present the syntax and the semantics of our model of 'recursive definitions'. We give formal definitions of the concepts of fixpoints and computed functions of such 'recursive definitions'.

2.2 Syntax

2.2.1 Alphabet

An alphabet, denoted $A(F,\bar{x})$, is a collection of symbols partitioned in the following way:

(i) There are n symbols, denoted x_i, $1 \leq i \leq n$, which are called individual variables. We denote the n-tuple $\langle x_1, x_2, \ldots, x_n \rangle$ by $\bar{x}$.

(ii) There is one symbol, denoted F, which is called function variable.

(iii) There is a non-empty set C of symbols called constant symbols.

(iv) There is one special symbol, denoted $\underset{\sim}{\perp}$, which is called the undefined symbol.

(v) The other symbols are called given function symbols. The set of given function symbols is partitioned into classes $G_1, G_2, \ldots, G_p, \ldots$ where, for every positive integer p, the elements of G_p are called the p-ary given function symbols. Some or all the G 's may be empty.

The symbols (iii), (iv), (v) are denoted by small letters with an underline $\sim$: $\underset{\sim}{a}, \underset{\sim}{b}, \underset{\sim}{c}, \ldots$ for the constants, $\underset{\sim}{\perp}$ for the undefined symbol, $\underset{\sim}{g}, \underset{\sim}{h}, \underset{\sim}{k}, \ldots$ for the functions.

At this point, we emphasize that the elements in the alphabet are just symbols, and do not have any meanings as variables or functions of some sort, in spite of the way they are called.

2.2.2 Terms

The terms over an alphabet are defined inductively in the following way:

(i) The individual variables x_i, for $1 \leq i \leq n$, are terms.

(ii) The constants in C are terms.

(iii) $\mathfrak{L}$ is a term.

(iv) For every $p > 0$, and every $\underset{\sim}{g} \in G_p$ if $\tau_1, \tau_2, \ldots, \tau_p$ are terms, then $\underset{\sim}{g}(\tau_1, \tau_2, \ldots, \tau_p)$ is a term.

(v) If $\tau_1, \ldots, \tau_n$ are terms, $F(\tau_1, \ldots, \tau_n)$ is a term.

(vi) There are no other terms than those which can be obtained from (i), (ii) and (iii) by a finite number of applications of (iv) and (v).

For example, if $\underset{\sim}{c} \in C$, $\underset{\sim}{g}_1 \in G_1$, $\underset{\sim}{g}_2 \in G_2$, and $n = 1$, $\underset{\sim}{g}_2(\underset{\sim}{g}_1(F(x)), \underset{\sim}{g}_2(\underset{\sim}{c}, \mathfrak{L}))$ is a term.

Terms are usually denoted by greek letters $\alpha, \beta, \ldots \sigma, \tau, \ldots$. Sometimes, when we wish to emphasize the variables of the alphabet, we denote the terms $\alpha(F, \bar{x})$, $\beta(F, \bar{X})$, etc... .

For proving properties of terms, we will frequently use what is known as the principle of structural induction, which is valid here because of the extremal clause (vi). It is a special case of Noetherian induction (see Cohn (1965)) and has been discussed thoroughly in Burstall (1969). Let us briefly recall here some of the relevant notations and facts.

We define an ordering relation between the terms over an alphabet in the following obvious way, by first defining the "immediate subterm" relation:

Definition A term α is an immediate subterm of a term β if and only if:

(i) $\beta = \underset{\sim}{g}(\tau_1, \ldots, \tau_p)$ for some $p > 0$, some $\underset{\sim}{g} \in G_p$ and $\alpha = \tau_i$ for some i, $1 \leq i \leq p$,

or: (ii) $\beta = F(\tau_1, \ldots, \tau_n)$ and $\alpha = \tau_i$ for some i, $1 \leq i \leq n$.

Definition The "proper subterm" relation is the transitive (non reflexive) closure of the "immediate subterm" relation.

The "proper subterm" relation is a (strict) ordering relation between terms, in which every decreasing chain has a minimal element (because of the extremal clause 2.2.2 vi). Structural induction is therefore a valid principle over the set of terms structured by the "proper subterm" relation, i.e., to prove that a property Φ holds for every term, it is sufficient to show that:

(i) For every i, $1 \leq i \leq n$, $\Phi(x_i)$.

(ii) For every $\underset{\sim}{c} \in C$, $\Phi(\underset{\sim}{c})$.

(iii) $\Phi(\underset{\sim}{x})$.

(iv) For every $p > 0$, for every p-tuple of terms $\tau_1,\ldots,\tau_p$, and for every $\underset{\sim}{g} \in G_p$, if $\Phi(\tau_1)$ and $\Phi(\tau_2)$ and...and $\Phi(\tau_p)$, then $\Phi(\underset{\sim}{g}(\tau_1,\ldots,\tau_p))$.

(v) For every n-tuple of terms $\tau_1,\ldots,\tau_n$, if $\Phi(\tau_1)$ and $\Phi(\tau_2)$ and... and $\Phi(\tau_n)$ then $\Phi(F(\tau_1,\ldots,\tau_n))$.

2.2.3 Substitutions

Substituting a term α for a term β in a term τ can yield different results according to which and how many occurrences of β in τ are substituted for. Let $S^{\alpha}_{\beta}\tau$ denote the set of all possible terms obtained by replacing β by α in τ. By convention, it will contain τ itself (corresponding to substitution of α for no occurrence of β in τ). $S^{\alpha}_{\beta}\tau$ can be defined inductively in an obvious way:

(a) if $\beta = \tau$, then $S^{\alpha}_{\beta}\tau = \{\alpha,\tau\}$;

(b) otherwise ($\beta \neq \tau$):

(b-i) - (biii) if τ is an individual variable x_i, $1 \leq i \leq n$, or if τ is a constant in C, or if τ is $\underset{\sim}{x}$, then: $S^{\alpha}_{\beta}\tau = \{\tau\}$.

(b-iv) if $\tau = \underset{\sim}{g}(\tau_1,\ldots,\tau_p)$, then:

$$S^{\alpha}_{\beta}\,\tau = \{\underset{\sim}{g}(\sigma_1,\ldots,\sigma_p) \mid \sigma_1 \in S^{\alpha}_{\beta}\,\tau_1,\ldots,\sigma_p \in S^{\alpha}_{\beta}\,\tau_p\}.$$

(b-v) if $\tau = F(\tau_1,..,\tau_n)$, then:

$$S^{\alpha}_{\beta}\,\tau = \{F(\sigma_1,..\ \sigma_n) \mid \sigma_1 \in S^{\alpha}_{\beta}\,\tau_1,\ldots,\sigma_n \in S^{\alpha}_{\beta}\,\tau_n\}.$$

Remark Such an inductive definition defines uniquely the set $S^{\alpha}_{\beta}\,\tau$ for any triple of terms α, β, τ, as can be shown rigorously by structural induction on τ. The detailed proof is left to the reader. We will very often use similar inductive definitions in the remainder of this work, without proving that such definitions indeed uniquely define what they are supposed to: the proof is generally trivial by structural induction.

2.3 Semantics

In this section we define the semantics of our system, that is we define interpretations of the alphabet and of the terms.

2.3.1 Interpretation of an alphabet

An interpretation of an alphabet consists of:

(a) A non-empty domain Δ, together with a 1-1 mapping of C (the set of constants of the alphabet) onto Δ. If $\underset{\sim}{c}$ is a constant in C, we denote its image in Δ by this mapping by c, and we call c the value of $\underset{\sim}{c}$. Conversely $\underset{\sim}{c}$ can be thought of as the name of c. The constraint for the mapping to be onto implies that $|\,C\,| = |\,\Delta\,|$ and says that every element of Δ has a name, which will be convenient later.

(b) An element ω not in Δ, which may be thought of as the 'undefined' value (i.e. the value of the 'undefined symbol' $\underset{\sim}{\omega}$).

(c) A subset D of Δ. (The motivation for introducing D has been given in the general discussion, § 1.3.2).

(d) For every $p \geq 1$ and for every $\underset{\sim}{g} \in G_p$, the interpretation (or value)

of $\underset{\sim}{g}$ will be a mapping g of a non empty subset of $(\Delta^+)^p$, called Dom(g), into Δ^+ */. These mappings g are actually the base functions (or given functions) of the interpretation. We refer the reader to the discussion in § 1.3.2. of these base functions, and to the definition of the sequential and parallel 'if ... then ... else' as examples of such functions.

Note that an element of the alphabet is represented by an underlined letter, whereas its interpretation is represented by the same letter without underline. We will omit the underline whenever no confusion can arise. For example, according to our notations, the correct way of writing the term used in the familiar recursive definition of the factorial over the integers is:

$\underset{\sim}{\text{if}}\ x \underset{\sim}{=} 0\ \underset{\sim}{\text{then}}\ 1\ \underset{\sim}{\text{else}}\ x \underset{\sim}{*} F(x \underset{\sim}{-} 1)$.

One adequate interpretation is the following:

The domain D is the set of integers, the domain Δ is $D \cup \{T,F\}$. Thus $\underset{\sim}{-3}$, $\underset{\sim}{0}$, $\underset{\sim}{1}$, $\underset{\sim}{T}$ are constant symbols, for example, and -3, 0, 1, T are their values.

Given Functions Symbols:	Interpretation:
$\underset{\sim}{\text{if}} \ldots \underset{\sim}{\text{then}} \ldots \underset{\sim}{\text{else}} \ldots$	The sequential 'if ... then ... else ...' defined in § 1.3.2.
$\underset{\sim}{=}$	The equality predicate = over $(D^+)^2$.
$\underset{\sim}{*}$	The multiplication * defined over $(D^+)^2$ by: $\forall\, x,y \in D$, $x*y$ is the usual product of x by y. $\forall\, x \in D$, $x * \omega = \omega * x = \omega * \omega = \omega$.
$\underset{\sim}{-}$	the subtraction - defined over $(D^+)^2$ by: $\forall\, x,y \in D$, $x - y$ is the usual subtraction

*/ Recall that, for any set S, we denote $S \cup \{\omega\}$ by S^+

of y from x

$$\forall x \in D,\ \omega - x = x - \omega = \omega - \omega = \omega$$

When we do not wish to emphasize the distinction between syntactic terms and their values, we simply write the same term:

$$\underset{\sim}{\text{if}}\ x = 0\ \underset{\sim}{\text{then}}\ 1\ \underset{\sim}{\text{else}}\ x * F(x - 1)$$

2.3.2. Evaluation of terms

Interpretations of the alphabet leave F and the x_i's free. If we supply some "value" to the functional variable F, as a partial function over some domain, and some "values" to the individual variables x_i's, it becomes possible to evaluate the terms */ over an alphabet $A(F,\bar{x})$.

More precisely, there are two natural ways of specifying F :

(a) One way is to specify F as a partial function of D^n into D (i.e. a mapping of D^n into D^+).

(b) The other is to specify F as a partial function of $(D^+)^n$ into D (i.e. a mapping of $(D^+)^n$ into D^+)**/

---

*/ Here and in most places throughout this work, we use the word "term" to mean "interpreted term", as there is usually no possible confusion.

**/ See Appendix I for a presentation of the main results regarding partial functions. Here are some definitions and notations that will be used throughout this work:

A partial function of a domain S into a range R is a mapping of S into R^+. We denote the set of such partial functions $pf(S \to R)$.

For convenience, we abbreviate the set of partial functions of D^n into D, $pf(D^n \to D)$, by $pf_n(D)$, and the set of partial functions of $(D^+)^n$ into D, $pf((D^+)^n \to D)$, by $pf_n(D^+)$.

In both cases, x_i can be specified as an element of $\Delta^+ \cup \{d\}$, where d is an element not in Δ and $\neq \omega$. (The meaning of d is discussed below. There are technical reasons, which will be apparent in some proofs later on, why one wants to define the evaluation of terms for x_i's outside D^+).

Corresponding to these two ways of specifying the variables, there are two ways of evaluating a term, which yield what we call, respectively, the <u>weak value</u> and the <u>strong value</u> of the term:

(a) the <u>weak value</u> of a term $\tau = \tau(F,\bar{x})$ for $F = f \in pf_n(D)$ and $\bar{x} = \bar{\xi} \in (\Delta^+ \cup \{d\})^n$ is an element of $\Delta^+ \cup \{d\}$, denoted $\tilde{\tau}(f,\bar{\xi})$, which is defined inductively as follows:

(i) if $\tau = x_i$, $\tilde{\tau}(f,\bar{\xi}) \equiv \xi_i$;

(ii) if $\tau = \underset{\sim}{c}$, where $\underset{\sim}{c}$ is a constant in C, $\tilde{\tau}(f,\bar{\xi}) \equiv c$;

(iii) if $\tau = \omega$, $\tilde{\tau}(f,\bar{\xi}) \equiv \omega$;

(iv) if $\tau = \underset{\sim}{g}(\tau_1, \tau_2, \ldots, \tau_p)$ then:

- if the vector $\langle \tilde{\tau}_1(f,\bar{\xi}), \ldots, \tilde{\tau}_p(f,\bar{\xi}) \rangle$ does not belong to Dom(g) then $\tilde{\tau}(f,\bar{\xi}) \equiv d$;
- otherwise, $\tilde{\tau}(f,\bar{\xi}) \equiv g(\tilde{\tau}_1(f,\bar{\xi}), \ldots, \tilde{\tau}_p(f,\bar{\xi}))$;

(v) if $\tau = F(\tau_1, \ldots, \tau_n)$ then:

- if the vector $\langle \tilde{\tau}_1(f,\bar{\xi}), \ldots, \tilde{\tau}_n(f,\bar{\xi}) \rangle$ does not belong to $(D^+)^n$ then $\tilde{\tau}(f,\bar{\xi}) \equiv d$;
- otherwise, if for some i, $1 \leq i \leq n$, $\tilde{\tau}_i(f,\bar{\xi}) \equiv \omega$, then $\tilde{\tau}(f,\bar{\xi}) \equiv \omega$;
- otherwise $\tilde{\tau}(f,\bar{\xi}) \equiv f(\tilde{\tau}_1(f,\bar{\xi}), \ldots, \tilde{\tau}_n(f,\bar{\xi}))$.

Again, as indicated in § 2.2.4, an easy proof by structural induction shows that (i) - (v) uniquely define $\tilde{\tau}(f,\bar{\xi})$ for

any term τ, any $f \in pf_n(D)$ and any $\xi \in (\Delta^+ \cup \{d\})^n$.

Note that the 'd' symbol that we introduce here corresponds to a 'don't care' condition, or, better, an 'error message' reported by the evaluation mechanism. It has a very different intuitive meaning from the 'ω' symbol which corresponds to a loop.

(b) The <u>strong value</u> of a term $\tau(F,\bar{x})$ for $F = f \in pf_n(D^+)$ and $\bar{x} = \bar{\xi} \in (\Delta^+ \cup \{d\})^n$ is an element of $\Delta^+ \cup \{d\}$, denoted $\tilde{\tilde{\tau}}(f,\bar{\xi})$, which is defined inductively as follows:

(i) - (iv) as in (a);

(v) if $\tau = F(\tau_1,\ldots,\tau_n)$ then:

- if the vector $\langle \tilde{\tilde{\tau}}_1(f,\bar{\xi}),\ldots,\tilde{\tilde{\tau}}_n(f,\bar{\xi})\rangle$ does not belong to $(D^+)^n$ then $\tilde{\tilde{\tau}}(f,\bar{\xi}) \equiv d$;
- otherwise, $\tilde{\tilde{\tau}}(f,\bar{\xi}) \equiv f\,(\tilde{\tilde{\tau}}_1(f,\bar{\xi}),\ldots,\tilde{\tilde{\tau}}_n(f,\bar{\xi}))$.

The only difference between the two evaluations resides in the evaluation of term of the form $F(\tau_1,\ldots,\tau_n)$. In the weak evaluation, since f is a mapping of D^n into D^+, $f(\xi_1,\ldots\xi_n)$ has no obvious meaning if one of the ξ_1's is ω. The solution adopted in a-(v) is to arbitrarily set $f(\xi_1,\ldots,\xi_n) = \omega$ whenever one or more of the arguments is ω. In the strong evaluation such a problem does not arise, since f is specified as a mapping of $(D^+)^n$ into D^+, and therefore $f(\xi_1,..,\xi_n)$ is known for all combinations of undefined arguments. Paragraph 2.3.5 will discuss the relations between the two kinds of evaluations.

We will use later the concepts of correct and compatible terms.

<u>Definition</u>: A term $\tau(F,\bar{x})$ is <u>correct</u> iff:

$\forall f \in pf_n(D^+)$, $\forall \bar{\xi} \in (D^+)^n$, $\tilde{\tilde{\tau}}(f,\bar{\xi}) \in \Delta^+$.

In other words, the strong values of τ always belong to Δ^+. Notice that every subterm of correct term is correct.

Definition. A term $\tau(F,\bar{x})$ is compatible iff:

$$\forall f \in pf_n(D^+),\ \forall \bar{\xi} \in (D^+)^n,\ \tilde{\tau}(f,\bar{\xi}) \in D^+.$$

In other words the strong values of τ always belong to D^+.

2.3.3 Functionals associated with a term.

Let τ be a correct term, as defined in the previous section. Corresponding to the two ways of evaluating τ, there are two functionals that can be associated with τ:

(a) $\tilde{\tau}$:

Corresponding to the weak evaluation we associate a functional, denoted $\tilde{\tau}$, which maps any partial function $f \in pf_n(D)$ into a partial function $\tilde{\tau}[f] \in pf(D^n \to \Delta)$ defined by:

$$\forall \bar{\xi} \in D^n: \tilde{\tau}[f](\bar{\xi}) = \tilde{\tau}(f,\bar{\xi})^{\prime}.$$

(b) $\tilde{\tilde{\tau}}$:

Corresponding to the strong evaluation we associate a functional, denoted $\tilde{\tilde{\tau}}$, which maps any partial function $f \in pf_n(D^+)$ into a partial function $\tilde{\tilde{\tau}}[f] \in pf((D^+)^n \to \Delta)$, defined by:

$$\forall \bar{\xi} \in (D^+)^n: \tilde{\tilde{\tau}}[f](\bar{\xi}) = \tilde{\tilde{\tau}}(f,\bar{\xi}).$$

If, in addition, τ is compatible, then $\tilde{\tau}$ is a functional over $pf_n(D)$, (i.e. a mapping of $pf_n(D)$ into $pf_n(D)$), and $\tilde{\tilde{\tau}}$ is a functional over $pf_n(D^+)$.

---

[1] The fact that $\tilde{\tau}[f] \in pf(D^n \to \Delta)$ when τ is correct comes from the fact that τ correct implies:

for any $f \in pf_n(D)$, for any $\bar{\xi} \in D^n$, $\tilde{\tau}(f,\bar{\xi}) \in \Delta^+$, and from Lemma 2.3.5.1. below.

2.3.4 'Recursive definitions' and their fixpoints.

We call 'recursive definition' an expression of the form

$$F(\bar{x}) \Leftarrow \tau(F,\bar{x})$$

where $\tau(F,\bar{x})$ is any compatible term, as defined in Paragraph 2.3.2.

Corresponding to the two functionals associated with τ, one can define two different classes of fixpoints of a 'recursive definition':

(a) A partial function f in $pf_n(D)$ is a weak fixpoint of the 'recursive definition' $F(\bar{x}) \Leftarrow \tau(F,\bar{x})$ iff it is a fixpoint of τ, i.e.: $\forall \bar{\xi} \in D^n: f(\bar{\xi}) = \tau(f,\bar{\xi})$.

(b) A partial function f in $pf_n(D^+)$ is a strong fixpoint of the 'recursive definition' $F(\bar{x}) \Leftarrow \tau(F,\bar{x})$ iff it is a fixpoint of $\tilde{\tau}$, i.e.: $\forall \bar{\xi} \in (D^+)^n: f(\bar{\xi}) = \tilde{\tau}(f,\bar{\xi})$.

The following section discusses in particular relations between the two kinds of fixpoints.

2.3.5 Relations between the two types of evaluations and fixpoints.

Let us denote by f^+ the natural extension of a partial function $f \in pf_n(D)$, i.e. f^+ is defined by:

$$\begin{cases} \forall \bar{\xi} \in D^n : \quad f^+(\bar{\xi}) \equiv f(\bar{\xi}) \\ \forall \bar{\xi} \in (D^+)^n \setminus (D)^n : f^+(\bar{\xi}) \equiv \omega \end{cases}.$$

Then we have:

2.3.5.1 Lemma: For every partial function $f \in pf_n(D)$, for every $\bar{\xi} \in D^n$, for every term $\tau(F,\bar{x})$:

$$\tilde{\tau}(f,\bar{\xi}) \equiv \tilde{\tilde{\tau}}(f^+,\bar{\xi}).$$

Proof: By structural induction on τ.

(i) - (iii) $\tau = x_i$, $\tau = \underset{\sim}{c}$, $\tau = \underset{\sim}{w}$: trivial.

(iv) $\tau = \underset{\sim}{g}(\tau_1,\ldots,\tau_p)$. By induction hypothesis, the vectors $\langle \tilde{\tau}_1(f,\bar{\xi}),\ldots,\tilde{\tau}_p(f,\bar{\xi})\rangle$ and $\langle \tilde{\tilde{\tau}}_1(f^+,\bar{\xi}),\ldots,\tilde{\tilde{\tau}}_p(f^+,\bar{\xi})\rangle$ are equal. Hence if they do not belong to $Dom(g)$, $\tilde{\tau}(f,\bar{\xi}) \equiv \tilde{\tilde{\tau}}(f^+,\bar{\xi}) \equiv d$,

and if they do:

$$\begin{aligned}\tilde{\tau}(f,\bar{\xi}) &\equiv g(\tilde{\tau}_1(f,\bar{\xi}),\ldots,\tilde{\tau}_p(f,\bar{\xi}))\\ &\equiv g(\tilde{\tilde{\tau}}_1(f^+,\bar{\xi}),\ldots,\tilde{\tilde{\tau}}_p(f^+,\bar{\xi}))\\ &\equiv \tilde{\tilde{\tau}}(f^+,\bar{\xi}).\end{aligned}$$

(v) $\tau = F(\tau_1,\ldots,\tau_n)$:

Again by induction hypothesis, the vectors $\langle \tilde{\tau}_1(f,\bar{\xi}),\ldots,\tilde{\tau}_p(f,\bar{\xi})\rangle$ and $\langle \tilde{\tilde{\tau}}_1(f^+,\bar{\xi}),\ldots,\tilde{\tilde{\tau}}_p(f^+,\bar{\xi})\rangle$ are equal. If they do not belong to $(D^+)^n$, then $\tilde{\tau}(f,\bar{\xi}) \equiv \tilde{\tilde{\tau}}(f^+,\bar{\xi}) \equiv d$.

If they do, then two cases arise:

(1) For some i, $1 \leq i \leq n$, $\tilde{\tau}_i(f,\bar{\xi}) \equiv \tilde{\tilde{\tau}}_i(f^+,\bar{\xi}) \equiv w$.

Then:

$$\begin{aligned}\tilde{\tilde{\tau}}(f^+,\bar{\xi}) &= f^+(\tilde{\tilde{\tau}}_1(f^+,\bar{\xi}),\ldots,\tilde{\tilde{\tau}}_n(f^+,\bar{\xi})) && \text{definition of } \tilde{\tilde{\tau}}\\ &\equiv w && \text{definition of } f^+\\ &\equiv \tilde{\tau}(f,\bar{\xi}) && \text{definition of } \tilde{\tau}\end{aligned}$$

(2) For every i, $1 \leq i \leq n$, $\tau_i(f,\bar{\xi}) = \tilde{\tau}_i(f^+,\bar{\xi}) \neq \omega$.

Then $\langle \tilde{\tau}_1(f,\bar{\xi}),\ldots,\tilde{\tau}_n(f,\bar{\xi}) \rangle \in D^n$ and we have:

$$\begin{aligned}
\tilde{\tau}(f^+,\bar{\xi}) &\equiv f^+(\tilde{\tau}_1(f^+,\bar{\xi}),\ldots,\tilde{\tau}_n(f^+,\bar{\xi})) && \text{definition of } \tilde{\tau}\\
&\equiv f(\tilde{\tau}_1(f^+,\bar{\xi}),\ldots,\tilde{\tau}_n(f^+,\bar{\xi})) && \text{definition of } f^+\\
&\equiv f(\tilde{\tau}_1(f,\bar{\xi}),\ldots,\tilde{\tau}_n(f,\bar{\xi})) && \text{induction hypothesis}\\
&\equiv \tilde{\tau}(f,\bar{\xi}) && \text{definition of } \tau
\end{aligned}$$

□

However there is in general no interesting relation between the fixpoints of τ and the fixpoints of $\tilde{\tau}$ as we show below.

2.3.5.2 A 'recursive definition' can have a weak fixpoint and no strong fixpoint.

Example: Let us take, $F(x) = \tau(F,x)$ where $\tau(F,x) = \underset{\sim}{g}(F(\underset{\sim}{\omega}))$.

Interpretation: $D = \Delta =$ {integers}

$$g = \begin{cases} g(\xi) \equiv \omega \text{ if } \xi \in D \\ g(\omega) \equiv 1 \,. \end{cases}$$

Now $f \in pf(D)$, defined by: $\forall \xi \in D, f(\xi) \equiv 1$, is a fixpoint of τ, hence a weak fixpoint of the 'recursive definition'.

However, this 'recursive definition' has no strong fixpoint, for suppose $\varphi \in pf(D^+)$ is such a fixpoint. Then:

$$\begin{aligned}
\forall \xi \in D^+ \quad \varphi(\xi) &\equiv \tilde{\tau}(\varphi,\xi) && \text{definition of a strong fixpoint}\\
&\equiv g(\varphi(\omega)) \,. && \S\ 2.3.2\ (b)
\end{aligned}$$

Hence, we have: $\varphi(\omega) = g(\varphi(\omega))$.

Now if $\varphi(\omega) \equiv \omega$ we get $\varphi(\omega) \equiv g(\omega) \equiv 1$: contradiction ;

if $\varphi(\omega) \not\equiv \omega$ we get $\varphi(\omega) \equiv \omega$: contradiction .

□

2.3.5.3 Conversely, a recursive definition can have a strong fixpoint and no weak fixpoint.

Example: Let us consider $F(x) \Leftarrow \tau(F,x)$, where:

$$\tau(F,x) = \underset{\sim}{h}\,(F(\underset{\sim}{\omega}),F(x)) .$$

Interpretation: $D = \Delta = \{\text{non negative integers}\}$. Let $Dom(h) = D^+ \times D^+$,

$$h = \begin{cases} h(\omega,\omega) \equiv 0 \\ h(\omega,\xi) \equiv \omega \quad \text{if } \xi \in D \\ h(\xi,\tau) \equiv 0 \quad \text{otherwise.} \end{cases}$$

Then, $\varphi \in pf(D^+)$ defined by: $\forall\, \xi \in D^+$, $\varphi(\xi) \equiv 0$ is a fixpoint of $\tilde{\tau}$, since:

$\forall\, \xi \in D^+$: $\tilde{\tau}(\varphi,\xi) \equiv h(\varphi(\omega), \varphi(\xi))$ § 2.3.2 (b)

$\equiv h(0,0)$ definition of φ

$\equiv 0$ definition of h

$\equiv \varphi(\xi)$ definition of φ .

However, this 'recursive definition' has no weak fixpoint for suppose $\psi \in pf(d)$ is such a fixpoint. Then:

$\forall\, \xi \in D$, $\psi(\xi) \equiv h(\omega,\psi(\xi))$ § 2.3.2 (a) .

- if $\psi(\xi) \equiv \omega$, then $\omega \equiv h(\omega,\omega) \equiv 0$: contradiction ;
- if $\psi(\xi) \in D$ then $\psi(\xi) \equiv \omega$: contradiction .

□

2.3.5.4 Finally, a 'recursive definition' can have a weak fixpoint and a strong fixpoint which agree nowhere on D, although they are both completely defined on D.

Ex: Let us consider the 'recursive definition' $F(x) \Leftarrow \tau(F,x)$, where $\tau(F,x) = \underset{\sim}{k}(F(\underset{\sim}{\omega}))$, with the following interpretation:

$D = \Delta = \{\text{integers}\}$. Let $Dom(k) = D^+$,

$$k = \begin{cases} k(\xi) = 0 & \text{if } \xi \in D \\ k(\Omega) = 1 \ . \end{cases}$$

Now the only fixpoint of $\tilde{\tau}$ is $\varphi_1 \in pf(D)$ defined by:

$$\forall \xi \in D, \quad \varphi_1(\xi) = 1 \ ,$$

and the only fixpoint of $\tilde{\tilde{\tau}}$ is $\varphi_2 \in pf(D^+)$ defined by:

$$\forall \xi \in D^+, \quad \varphi_2(\xi) = \Omega \ .$$

Thus, even though the definitions of $\tilde{\tau}$ and $\tilde{\tilde{\tau}}$ appear to be quite similar, the functionals $\tilde{\tau}$ and $\tilde{\tilde{\tau}}$ sometimes behave quite differently.

In the next section we define the computations of a 'recursive definition'.

2.4 Computations

Computations of 'recursive definitions' have been discussed in § 1.4.2. We will give here a more formal model.

2.4.1 Notations.

We first need to define the notation $\alpha(F,\bar{\beta})$ where $\bar{\beta}$ is a short notation for an n-tuple of terms $\beta_1,\beta_2,\ldots,\beta_n$. Informally, $\alpha(F,\bar{\beta})$ is the term obtained by substituting x_i by β_i for all occurrences of x_i in $\alpha(F,\bar{x})$, for every i, $1 \leq i \leq n$.

Formally, $\alpha(F,\bar{\beta})$ can be defined by structural induction in the obvious way:

(i) if $\alpha = x_i$, $1 \leq i \leq n$, then: $\alpha(F,\bar{\beta}) = \beta_i$;

(ii) if $\alpha = \underset{\sim}{a} \in C$ (i.e. constant term), then: $\alpha(F,\bar{\beta}) = \underset{\sim}{a}$;

(iii) if $\alpha = \underset{\sim}{x}$, then : $\alpha(F,\bar{\beta}) = \underset{\sim}{x}$;

(iv) if $\alpha = \underset{\sim}{g}(\alpha_1,\alpha_2,\ldots,\alpha_p)$, with $\underset{\sim}{g} \in G_p$, then:

$$\alpha(F,\bar{\beta}) = \underset{\sim}{g}(\alpha_1(F,\bar{\beta}),\ldots,\alpha_p(F,\bar{\beta})) \ ;$$

(v) if $\alpha = F(\alpha_1, \alpha_2, \ldots, \alpha_n)$, then:

$$\alpha(F, \bar{\beta}) = F(\alpha_1(F, \bar{\beta}), \ldots, \alpha_n(F, \bar{\beta})).$$

A frequent case is when $\bar{\beta}$ is an n-tuple of constants, i.e. $\beta_i = \underset{\sim}{c}_i$, with $c_i \in D^+$, for every i, $1 \leq i \leq n$.

In this case, we denote $\underset{\sim}{\bar{c}} = \langle \underset{\sim}{c}_1, \underset{\sim}{c}_2, \ldots, \underset{\sim}{c}_n \rangle$ and $\alpha(F, \underset{\sim}{\bar{c}})$ is defined as above with $\underset{\sim}{\bar{c}} = \bar{\beta}$.

2.4.2. Computations of $\alpha(F, \bar{x})$ for $\bar{x} = \bar{c}$

Definition: An elementary computation of a term $\alpha(F, \bar{x})$ over some interpreted alphabet, for $\bar{x} = \bar{c} \in (D^+)^n$, using a 'recursive definition' $F(\bar{x}) \Leftarrow \tau(F, \bar{x})$, is a sequence of terms $\{\alpha_i, i \geq 0\}$ such that:

(a) $\alpha_0 = \alpha(F, \underset{\sim}{\bar{c}})$ as defined in 2.4.1.

(b) For every $i \geq 0$,

- either (b1): $\alpha_{i+1} \in S^{\beta}_{\gamma} \alpha_i$, where β is free of $\bar{x}$,*/ and the equation:

$$\tilde{\tilde{\beta}}(f, \ ) = \tilde{\tilde{\gamma}}(f, \ ) \text{ holds for every } f \in pf_n(D^+),$$

---

*/ The notation $S^{\beta}_{\gamma} \alpha$ has been defined in § 2.2.4, and the notation $\tau(F, \bar{\beta})$ in §2.4.1.

A term β is said to be free of $\bar{x}$ if it contains no instances of x_j, for any j, $1 \leq j \leq n$. Therefore, if γ is a subterm of α_i, it is obviously free of $\bar{x}$.

If a term β is free of $\bar{x}$, $\tilde{\beta}(f, \bar{\xi})$ does not depend on $\bar{\xi}$, and we denote it by $\tilde{\beta}(f, \ )$; similarly $\tilde{\beta}(f, F)$ does not depend on $\bar{F}$, and we denote it by $\tilde{\beta}(f, \ )$.

- or b.): $\alpha_{i+1} \in S^{\tau(F,\bar{\rho})}_{F(\bar{\rho})}\ \alpha_i$.

Definition: A computation of a term $\alpha(F,\bar{x})$ over some interpreted alphabet, for $\bar{x} = \bar{c} \in (D^+)^n$, using a 'recursive definition' $F(\bar{x}) \Leftarrow \tau(F,\bar{x})$, is any sequence of terms $\{\alpha_i \mid i \geq 0\}$ which is a subsequence */ of an elementary computation $\{\beta_j \mid j \geq 0\}$ of $\alpha(F,\bar{x})$, for $\bar{x} = \bar{c}$ using the same 'recursive definition', and such that $\alpha_o = \beta_o$.

2.4.3 Terminating computations.

Let $\alpha(F,\bar{x})$ be a term over some interpreted alphabet and $a \in \Delta$ be a constant. A computation $\{\alpha_i, i \geq 0\}$ of α is said to terminate with $\underset{\sim}{a}$, if there is some n, $n \geq 0$ such that $\alpha_n = \underset{\sim}{a}$. A computation $\{\alpha_i, i \geq 0\}$ of α is said to terminate if there is an $a \in \Delta$ such that the computation terminates with $\underset{\sim}{a}$. Note that a is forbidden to be ω).

---

*/ By that, we mean that there is a mapping s of the nonnegative integers into the nonnegative integers such that:

(a) for all $i \geq 0$: $\beta_{s(i)} = \alpha_i$,

(b) for all $i, j \geq 0$: $i > j \Rightarrow s(i) > s(j)$.

The purpose of introducing these subsequences is to allow one to "skip steps" during a computation, or replace some term β by b if a computation of β is known to terminate with b, for example from previous computations. [Terminating computations are defined in § 2.4.3.]

Comments: Notice that according to our definition, a computation can "terminate" and be infinite at the same time. (In fact, technically, all computations are infinite). The point is that, for terminating computations, only a finite initial subsequence is considered significant.

2.4.4 Computed functions.

As emphasized in Chapter 1 there are usually many possible computations for a term. Computed functions of a 'recursive definition' are defined by considering all possible computations of $F(\bar{x})$ for $\bar{x}$ ranging through D^n or $(D^+)^n$. Let $\bar{x}$ be fixed to some $\bar{c}$ in D^n or $(D^+)^n$. Then it is possible to associate values to every computation of $F(\bar{x})$ for $\bar{x}$ being $\bar{c}$: If the computation does not terminate, then the associated value is ω. If the computation terminates with $\underset{\sim}{a} \in C$, then an associated value is a. [Note that, so far, there might be several values associated to a given computation]. A computed function is defined as being any partial function (in $pf_n(D)$ or $pf_n(D^+)$) mapping $\bar{c}$ (in D^n or $(D^+)^n$) into a value associated with some computation of $F(\bar{x})$ for $\bar{x} = \bar{c}$.

Another, equivalent, way of stating this is:

Definition: A partial function $f \in pf_n(D^+)$ [resp $\in pf_n(D)$] is said to be a computed function of a 'recursive

definition' over $(D^+)^n$ [resp over D^n] if it is such that, for every $\bar{c} \in (D^+)^n$ [resp $\in D^n$]:

(i) if $f(\bar{c}) \neq \Omega$ then there is a computation of $F(\bar{x})$ for $\bar{x} = \bar{c}$ using the 'recursive definition' which terminates with $\underset{\sim}{f}(\bar{c})$.

(ii) if $f(\bar{c}) = \Omega$ then there is a computation of $F(\bar{x})$ for $\bar{x} = \bar{c}$ using the 'recursive definition' which does not terminate.

Notice that, because of the way we have defined substitutions any non-constant term α has non-terminating computations: a trivial example is the sequence $\{\alpha_i \mid i \geq 0\}$ defined by $\alpha_i = \alpha_0$ for all $i \geq 0$, which is a non terminating computation [This is because, for any triplet of terms $\alpha, \beta, \gamma \in S^2_\beta$- from 2.4.4]. Consequently, for any specification of $\bar{x}$, $F(\bar{x})$ has non-terminating computations.

Hence if f and g are two partial functions such that $f \leq g$, and if g is a computed function, f is also a computed function.

2.4.5 Innermost computations.

We define innermost computations as computations in which (b2) of 2.4.2 is only applied when $\bar{\beta}$ is an n-tuple of individual constants in C. In other words, they are those computations in which only call by value is permitted.

In the course of Chapter 5, we will define other types of computations, as needed.

CHAPTER 3

RELATIONS BETWEEN COMPUTED FUNCTIONS AND FIXPOINTS

3.1 Introduction

3.2 Computed Functions and Strong Fixpoints

3.2.1 First Substitution Lemma for Strong Values

3.2.2 Second Substitution Lemma for Strong Values

3.2.3 Theorem 1

3.3 Innermost Computed Functions and Weak Fixpoints

3.3.1 First Substitution Lemma for Weak Values

3.3.2 Second Substitution Lemma for Weak Values

3.3.3 Theorem 2

3.1 Introduction

In this chapter, we essentially present detailed and complete proofs of Theorems 1 and 2 discussed in Section 1.5 of Chapter 1. It should be emphasized that there are no restrictions on the 'recursive definitions' to which these theorems apply. (For example, the base functions of the 'recursive definition' do not have to be monotonic.) They might not even be computable, and the results would still hold.

The proofs of Theorems 1 and 2 are very similar: We prove three Lemmas in Section 3.2 from which Theorem 1 follows easily. Then we proceed in an analogous fashion for Theorem 2, in Section 3.3.

3.2 Computed Functions and Strong Fixpoints

In this section we first give two lemmas which essentially express (semantic) stability properties of the strong value of a term when certain (syntactic) substitutions */ are performed in this term. We will use them to prove Theorem 1 in the last paragraph of this section.

3.2.1 First Substitution Lemma for Strong Values

For every triplet of terms α, β, τ, for every $f \in pf_n(D^+)$, for every $\bar{\xi} \in (\Delta^+ \cup \{d\})^n$, if $\tilde{\tilde{\alpha}}(f,\bar{\xi}) \equiv \tilde{\tilde{\beta}}(f,\bar{\xi})$, then for every $\sigma \in S^{\alpha}_{\beta}\tau$ the identity $\tilde{\tilde{\sigma}}(f,\bar{\xi}) \equiv \tilde{\tilde{\tau}}(f,\bar{\xi})$ holds.

Comment: Informally, this says that substituting one term for another within a third one (no matter how many occurrences are substituted for) does not alter the strong value of the latter term if the strong values of the two former ones are the same.

Proof: Let f be an arbitrary element of $pf_n(D^+)$, $\bar{\xi}$ an arbitrary n-tuple in $(\Delta^+ \cup \{d\})^n$, and α, β be two arbitrary terms such that

*/ Substitutions within terms have been formally defined in paragraph 2.2.4 (Chapter 2).

$\tilde{\tilde{\alpha}}(f,\bar{F}) \equiv \tilde{\tilde{\beta}}(f,\bar{F})$. We show that, for every τ and every σ in $S^{\alpha}_{\beta}\tau$ the identity $\tilde{\tilde{\sigma}}(f,\bar{F}) = \tilde{\tilde{\tau}}(f,\bar{F})$ holds. This is equivalent to the stated Lemma.

We first show as a separate case that this identity holds for $\tau = \beta$. Then, using structural induction, we show it for every τ.

Case 1: $\tau = \beta$

Then, by 2.2.4 (a), $S^{\alpha}_{\beta}\tau = \{\alpha,\tau\}$.

Therefore, either $\sigma = \tau$ in which case the result is trivial, or $\sigma = \alpha$ in which case:

$\tilde{\tilde{\sigma}}(f,\bar{F}) \equiv \tilde{\tilde{\alpha}}(f,\bar{F})$

$\equiv \tilde{\tilde{\beta}}(f,\bar{F})$ Hypothesis of the Lemma

$\equiv \tilde{\tilde{\tau}}(f,\bar{F})$ Hypothesis of Case 1.

Case 2: $\tau \neq \beta$

We now use structural induction on τ to show that for every term τ, $\tau \neq \beta$ implies that the property holds.

(i), (ii), and (iii) If τ is an individual variable x_i, $1 \leq i \leq n$, or if τ is $\underset{\sim}{y}$, or if τ is an individual constant in C, then $S^{\alpha}_{\beta}\tau = \{\tau\}$, from the definition in 2.2.4 (b).

Hence, $\sigma = \tau$ and the property is trivially true.

(iv) τ is of the form $\underset{\sim}{g}(\tau_1, \tau_2, \ldots, \tau_p)$. Let $\sigma \in S^{\alpha}_{\beta}\tau$.

Because $\tau \neq \beta$, case b-iv of the definition of substitutions in Section 2.2.4 applies, and we have:

$\sigma = \underset{\sim}{g}(\sigma_1, \sigma_2, \ldots, \sigma_p)$ where, for every i, $1 \leq i \leq p$, $\sigma_i \in S^{\alpha}_{\beta}\tau_i$.

Now, for every i, $1 \leq i \leq p$, we have:

$\tilde{\tilde{\sigma}}_i(f,\bar{F}) \equiv \tilde{\tilde{\tau}}_i(f,\bar{F})$,

either because of Case 1 if $\tau_i = \beta$, or because of the induction hypothesis otherwise.

Hence, we must have $\sigma(f,\bar{\xi}) = \tau(f,\bar{\xi})$ because of the definition of the strong values, Section 2.3.2 case b-iv.

(v) $\tau = F(\tau_1,\tau_2,\ldots,\tau_n)$.

In a similar way, let $\sigma \in S^{\alpha}_{\beta}\tau$. We have (2.2.4 b-v): $\sigma = F(\sigma_1,\sigma_2,\ldots,\sigma_n)$ where, for every i, $1 \leq i \leq n$, $\sigma_i \in S^{\alpha}_{\beta}\tau_i$. Again, for every i, $1 \leq i \leq n$, we have $\tilde{\sigma}_i(f,\bar{\xi}) \equiv \tilde{\tau}_i(f,\bar{\xi})$, and, by 2.3.2 b-v, this implies $\tilde{\sigma}(f,\bar{\xi}) \equiv \tilde{\tau}(f,\bar{\xi})$

□

3.2.2 Second Substitution Lemma for Strong Values

Let $\bar{\tau} = \langle \tau_1,\tau_2,\ldots,\tau_n \rangle$ be an arbitrary n-tuple of terms and α be any term. Then, for every $f \in pf_n(D^+)$, and for every $\bar{\xi} \in (\Delta^+ \cup \{d\})^n$, we have:

$$\widetilde{\alpha(F,\bar{\tau})}\,(f,\bar{\xi}) = \tilde{\alpha}(f, \langle \tilde{\tau}_1(f,\bar{\xi}),\ldots,\tilde{\tau}_n(f,\bar{\xi}) \rangle).$$ [*]

Comment: Informally, this property means that one can equivalently:

(a) First substitute τ_i for x_i everywhere in α, and then compute the strong value of the resulting term for f and $\bar{\xi}$,

or: (b) First compute the strong value of each τ_i for f and $\bar{\xi}$, and then compute the strong value of α for f and the vector of values of τ_i just computed.

---

[*] The notation $\alpha(F,\bar{\tau})$ has been formally defined in Paragraph 2.4.1: $\alpha(F,\bar{\tau})$ is the term obtained by substituting x_i by τ_i for all occurrences of x_i in α (for every i, $1 \leq i \leq n$).

Proof The proof is lengthy, but easy by structural induction on α.

Let us denote the vector $\langle \tilde{\tau}_1(f,\bar{\xi}),\ldots,\tilde{\tau}_n(f,\bar{\xi})\rangle$ by $\bar{t}$,

$\widetilde{\alpha(F,\bar{\tau})}\ (f,\bar{\xi})$ by l.h.s., and

$\tilde{\alpha}(f,\langle \tilde{\tau}_1(f,\bar{\tau}),\ldots,\tilde{\tau}_n(f,\bar{\tau})\rangle) \equiv \tilde{\alpha}(f,\bar{t})$ by r.h.s.

Case (i) $\alpha = x_i$ for some i, $1 \le i \le n$.

Then $\alpha(F,\bar{\tau}) \equiv \tau_i$, by 2.4.1 (i), and: l.h.s $\equiv \tilde{\tau}_i(f,\bar{\xi})$;

r.h.s. $\equiv \tilde{\tau}_i(f,\bar{\xi})$, by 2.3.2 (b-i), so l.h.s $\equiv$ r.h.s .

Case (ii) $\alpha = \underset{\sim}{c}$, for some $\underset{\sim}{c}$ in C.

Then $\alpha(F,\bar{\tau}) \equiv \underset{\sim}{c}$ by 2.4.1 (ii) and: l.h.s $\equiv c$ by 2.3.2 (b-ii) ;

r.h.s $\equiv c$, also by 2.3.2 (b-ii), so l.h.s $\equiv$ r.h.s .

Case (iii) $\alpha = \underset{\sim}{x}$.

In this case, l.h.s $\equiv$ r.h.s $\equiv x$ by an argument similar to case (ii).

Case (iv) $\alpha = \underset{\sim}{g}(\alpha_1,\alpha_2,\ldots,\alpha_p)$.

We have $\alpha(F,\bar{\tau}) \equiv \underset{\sim}{g}(\alpha_1(F,\bar{\tau}),\ldots,\alpha_p(F,\bar{\tau}))$, by 2.4.1-iv.

and so l.h.s $\equiv \widetilde{\underset{\sim}{g}(\alpha_1,\ldots,\alpha_p)(F,\bar{\tau})}\ (f,\bar{\xi})$

$\equiv \widetilde{\underset{\sim}{g}(\alpha_1(F,\bar{\tau}),\ldots,\alpha_p(F,\bar{\tau}))}\ (f,\bar{\xi})$.

Let us denote, for every i, $1 \le i \le p$, $\widetilde{\alpha_i(F,\bar{\tau})}\ (f,\bar{\xi})$ by a_i.

Then, by 2.3.2 (b-iv), if the vector $\langle a_1,a_2,\ldots,a_p\rangle$ does not belong to the domain of $\underset{\sim}{g}$, l.h.s $\equiv d$. If it does, l.h.s $\equiv$ $\underset{\sim}{g}(a_1,a_2,\ldots,a_p)$.

On the other hand r.h.s $\equiv \tilde{\gamma}(f,\bar{t})$

$\equiv \widetilde{\underset{\sim}{g}(\alpha_1,\alpha_2,\ldots,\alpha_p)}\ (f,\bar{t})$

Now, by induction hypothesis, for every i, $1 \leq i \leq p$, we have $\widetilde{\widetilde{\alpha}}_i(f,\bar{t}) \equiv \widetilde{\widetilde{\alpha_i(F,\bar{\tau})}}\ (f,\bar{\xi}) \equiv a_i$ as defined above.

Hence, by 2.3.2 (b-iv), if the vector $\langle a_1,a_2,\ldots,a_p\rangle$ does not belong to the domain of g, r.h.s $\equiv$ d. If it does, r.h.s $\equiv g(a_1,a_2,\ldots a_p)$.

So in both cases r.h.s $\equiv$ l.h.s.

Case (v) $\alpha = F(\alpha_1,\alpha_2,\ldots\alpha_n)$

The proof is very similar to the previous case. We have

$\alpha(F,\bar{\tau}) = F(\alpha_1(F,\bar{\tau}),\ldots,\alpha_n(F,\bar{\tau}))$ by 2.4.1-v, and so

l.h.s $\equiv \widetilde{\widetilde{\alpha(F,\bar{\tau})}}\ (f,\bar{\xi})$

$\equiv \widetilde{\widetilde{F(\alpha_1(F,\bar{\tau}),\ldots,\alpha_n(F,\bar{\tau}))}}\ (f,\bar{\xi})$

Let us again denote $\widetilde{\widetilde{\alpha_i(F,\bar{\tau})}}\ (f,\bar{\xi})$ by a_i, for every i, $1 \leq i \leq n$. By 2.3.2 (b-v), if the vector $\langle a_1,a_2,\ldots,a_n\rangle$ does not belong to $(D^+)^n$, then l.h.s $\equiv$ d. If it does, l.h.s $\equiv f(a_1,a_2,\ldots,a_n)$.

On the other hand, r.h.s $\equiv \widetilde{\widetilde{\alpha}}(f,\bar{t})$

$\equiv \widetilde{\widetilde{F(\alpha_1,\alpha_2,\ldots,\alpha_n)}}\ (f,\bar{t})$.

Now, by induction hypothesis, for every i, $1 \leq i \leq n$, we have $\widetilde{\widetilde{\alpha}}_i(f,\bar{\tau}) \equiv \widetilde{\widetilde{\alpha_i(F,\bar{\tau})}}\ (f,\bar{\xi}) \equiv a_i$ as defined above.

Hence, by 2.3.2 (b-v), if the vector $\langle a_1,a_2,\ldots,a_n\rangle$ does not belong to $(D^+)^n$, then r.h.s $\equiv$ d.

If it does, r.h.s $\equiv f(a_1,a_2,\ldots,a_n)$.

Again, in both cases, r.h.s $\equiv$ l.h.s.

□

3.2.3 Theorem 1

We now prove Theorem 1 which was stated in Section 1.5.

We need one preliminary Lemma:

Lemma: Let $\alpha(F,\bar{x})$ be a correct term over an interpreted alphabet, $\bar{c}$ be an element of $(D^+)^n$ and $F(\bar{x}) \Leftarrow \tau(F,\bar{x})$ be a recursive definition. Let $\{\alpha_i \mid i \geq 0\}$ be an arbitrary computation of α for $\bar{x} = \bar{c}$ using the recursive definition. Then, for every fixpoint φ of $\tilde{\tau}$, [*] and for all $i \geq 0$, $\tilde{\tilde{\alpha}}(\varphi,\bar{c}) = \tilde{\alpha}_i(\varphi,)$.

Proof: First remark that it is sufficient to show the property for elementary computations (defined in Section 2.4.2), since the general computations are subsequences of elementary ones.

Let φ be a fixpoint of $\tilde{\tau}$.

We begin by showing that, for every $i \geq 0$, if $\tilde{\alpha}_i(\varphi,) \in \Delta^+$, then

$$\tilde{\tilde{\alpha}}_i(\varphi,) = \tilde{\alpha}_{i+1}(\varphi,).$$

Because of the opening remark, we only have to consider two cases:

Case (a) $\alpha_{i+1} \in S^{\tau}_{\alpha_i}$

where, $\forall f \in pf_n(D^+)$, $\tilde{\varphi}(f,) = \tilde{\varphi}(f,)$.

In this case, a direct application of Lemma 3.2.1 proves the property.

Case (b) $\alpha_{i+1} \in S^{\tau(F,\bar{B})}_{F(\bar{B})} \alpha_i$.

In this case, we first observe that, if $F(\bar{B})$ is not present in α_i, then $\alpha_{i+1} = \alpha_i$ and the property is trivial. If it is present in α_i, then the hypothesis $\tilde{\alpha}_i(\varphi,) \in \Delta^+$ implies that, for every j, $1 \leq j \leq n$,

---

[*] $\tilde{\tau}$, functional associated with τ, has been defined in 2.4.2 (b).

$\tilde{\beta}_j(\varphi, \ ) \in D^+$. [Otherwise the subterm $F(\bar{\beta})$ would evaluate to 'd', and hence so would α_i as can be seen easily from the definitions in 2.3.2 b].

Therefore, we have:

$$\tilde{\tilde{\tau}}(F,\bar{\beta})\ (\varphi, \ ) \equiv \tilde{\tilde{\tau}}(\varphi, \langle \tilde{\beta}_1(\varphi, \ ), \ldots, \tilde{\beta}_n(\varphi, \ ) \rangle) \qquad \text{Lemma 3.2.2}$$

$$= \varphi(\tilde{\beta}_1(\varphi, \ ), \ldots, \tilde{\beta}_n(\varphi, \ )) \qquad \varphi \text{ is a fixpoint of } \tilde{\tilde{\tau}}$$

$$= \tilde{\tilde{F}}(\bar{\beta})\ (\varphi, \ ) \qquad 2.3.2 \text{ b-v}$$

And again, an immediate application of Lemma 3. .1 proves the property.

Now a trivial mathematical induction on i proves the Lemma, since $\tilde{\tilde{\alpha}}_0(\varphi, \ ) \equiv \tilde{\tilde{\alpha}}(\varphi,\bar{c})$, and i has been assumed correct, which implies $\tilde{\tilde{\alpha}}(\varphi,\bar{c}) \in \Delta^+$. □

We can now state and prove Theorem 1:

Theorem 1: For every 'recursive definition', every strong fixpoint is an extension $\dot{-}$ of every computed function over $(D^+)^n$.

Proof: Let $F(\bar{x}) \Leftarrow \tau(F,\bar{x})$ be a 'recursive definition', let f be one of its computed functions over $(D^+)^n$ and let φ be a fixpoint of $\tilde{\tilde{\tau}}$.

Let $\bar{\xi} \in (D^+)^n$ be such that $f(\bar{\xi}) \neq w$. Thus, from the definition of a computed function, we know that there is a computation $\{\alpha_i \mid i \geq 0\}$ such that $\alpha_0 = F(\bar{\xi})$ and $\alpha_k = f(\bar{\xi})$ for some finite k.

Now, by the previous Lemma, we know that, for every $i \geq 0$, $\tilde{\tilde{\alpha}}_i(\varphi, \ ) \equiv \tilde{\tilde{\alpha}}(\varphi,\bar{\xi}) \equiv \varphi(\bar{\xi})$.

In particular, we get that $\tilde{\tilde{\alpha}}_k(\varphi, \ ) \equiv \varphi(\bar{\xi})$. But since $\alpha_k = f(\bar{\xi})$, we see that $f(\bar{\xi}) \equiv \varphi(\bar{\xi})$, which is what we wanted.

□

---

*/ If f,g are partial functions of S into R, g is an extension of f iff: $\forall x \in S,\ f(x) \mid w \Rightarrow f(x) \equiv g(x)$. See Appendix I.

Some consequences of this Theorem have been stated and discussed in Section 1.5, and we will not go over them again here. They all follow immediately from the Theorem.

3.3 Innermost Computed Functions and Weak Fixpoints

In this section we derive the analogous of Lemma 3.2.1 and 3.2.2 for weak values, and then we prove Theorem 2.

3.3.1 First Substitution Lemma for Weak Values

For every triplet of terms α, β, τ, for every $f \in pf_n(D)$, for every $\bar{\xi} \in \Delta^+ \cup \{d\})^n$, if $\tilde{\alpha}(f,\bar{\xi}) \equiv \tilde{\beta}(f,\bar{\xi})$, then for every $\sigma \in S^{\alpha}_{\beta}\tau$, the identity $\tilde{\sigma}(f,\bar{\xi}) \equiv \tilde{\tau}(f,\bar{\xi})$ holds.

Proof: The proof of Lemma 3.2.1 carries over completely by simply changing '$\approx$' by '$\sim$', '2.3.2(b)' by '2.3.2(a)' 'D^+' by 'D', and 'strong' by 'weak'. □

3.3.2 Second Substitution Lemma for Weak Values

Let $\bar{\tau} = \langle \tau_1, \tau_2, \ldots, \tau_n \rangle$ be an arbitrary n-tuple of terms, and α be any term. Then for every $f \in pi_n(D)$ and for every $\bar{\xi} \in (\Delta^+ \cup \{d\})^n$, we have:

$$\widetilde{\alpha(F,\bar{\tau})}\ (f,\bar{\xi}) \equiv \tilde{\alpha}(f, \langle \tilde{\tau}_1(f,\bar{\xi}), \ldots, \tilde{\tau}_n(f,\bar{\xi}) \rangle).$$

Proof: The proof is by induction on α and parallels that of Lemma 3.2.2. Let us again abbreviate the left hand side and the right hand side of the identity which we want to prove by l.h.s. and r.h.s. respectively. Cases (i) through (iv) translate directly and we won't repeat the derivations here.

Case (v): $\alpha = F(\alpha_1, \alpha_2, \ldots, \alpha_n)$

We have $\alpha(F,\bar{\tau}) = F(\alpha_1(F,\bar{\tau}), \ldots, \alpha_n(F,\bar{\tau}))$ by 2.4.1-v,

and so:

$$\text{l.h.s} \equiv \widetilde{\alpha(F,\bar{\tau})}\ (f,\bar{\xi})$$

$$\equiv \widetilde{F(\alpha_1(F,\bar{\tau}),\ldots,\alpha_n(F,\bar{\tau}))}\ (f,\bar{\xi})\ .$$

Let us denote $\widetilde{\alpha_i(F,\bar{\tau})}\ (f,\bar{\xi})$ by a_i, for every i, $1 \leq i \leq n$. By induction hypothesis, we also have $\widetilde{\alpha}_i(f,\bar{t}) = a_i$. Three subcases arise:

v-(i): $\langle a_1,a_2,\ldots,a_n\rangle$ does not belong to $(D^+)^n$.

Then, by 2.3.2(a-v), l.h.s. $\equiv$ d.

On the other hand, r.h.s. $\equiv \widetilde{\alpha}(f,\bar{t})$

$$\equiv \widetilde{F(\alpha_1,\alpha_2,\ldots,\alpha_n)}\ (f,\bar{t})$$

$\equiv$ d, by 2.3.2(b-v) .

v-(ii): For some i, $1 \leq i \leq n$, $a_i \equiv \omega$.

Then, by 2.3.2(a-v), l.h.s. $\equiv \omega$.

On the other hand, r.h.s. $\equiv \omega$ by 2.3.2(a-v).

v-(iii): Otherwise, $(a_1,a_2,\ldots,a_n) \in D^n$ and we have:

l.h.s. $\equiv f(a_1,a_2,\ldots,a_n)$ by 2.3.2(a-v)

r.h.s. $\equiv f(a_1,a_2,\ldots,a_n)$ by 2.3.2(a-v). □

3.3.3 Theorem 2

In this section, we prove Theorem 2, which was stated in Section 1.5. We need the analogous of the Lemma in §3.2.3.

Lemma:

Let $\alpha(F,\bar{x})$ be a term over an interpreted alphabet, $\bar{c}$ be an element of D^n and $F(\bar{x}) \Leftarrow \tau(F,\bar{x})$ be a 'recursive definition'. Let $\{\alpha_i \mid i \geq 0\}$ be an arbitrary innermost computation of α for $\bar{x} = \bar{c}$ using the 'recursive definition'. Then for every fixpoint φ of $\widetilde{\tau}$, and for every $i \geq 0$:

$$\widetilde{\alpha}(\varphi,\bar{c}) \equiv \widetilde{\alpha}_i(\varphi,\ ) \quad .$$

Proof: Again the proof closely parallels that of Lemma 3.2.3. Notice that we do not need the correctness of α here. The same opening remark applies, that is, it is sufficient to show the property for elementary innermost computations.

Let φ be a fixpoint of $\tilde{\tau}$. Again, we first show that, for every $i \geq 0$, $\tilde{\alpha}_i(\varphi, \ ) \equiv \tilde{\alpha}_{i+1}(\varphi, \ )$.

Only two cases arise:

Case (a): $\alpha_{i+1} \in S^{\beta}_{\gamma} \alpha_i$, where:

$\forall\ f \in pf_n(D^+)$, $\tilde{\beta}(f, \ ) = \tilde{\gamma}(f, \ )$.

In particular, taking $f = \varphi^+$ (where φ^+ is the natural extension of φ, defined in Paragraph 2.3.5), we have:

$\tilde{\beta}(\varphi^+, \ ) \equiv \tilde{\gamma}(\varphi^+, \ )$.

Now, using Lemma 2.3.5.1, we get that:

$\tilde{\beta}(\varphi, \ ) = \tilde{\gamma}(\varphi, \ )$,

and using Lemma 3.3.1, we obtain the desired result*/

Case (b):

$\alpha_{i+1} \in S^{\tau(F,\bar{b})}_{F(\bar{b})} \alpha_i$

where $\underset{\sim}{\bar{b}}$ is an n-tuple of constants $\underset{\sim}{b}_i$ in C.

---

*/ Note that, in fact we could use a weaker condition to define "innermost computations over D^n", by only requiring that the weak values of β and γ be equal in 2.4.2(b1), instead of the strong values. The resulting Theorem 2 would be slightly stronger, since it would apply to a slightly larger class of computed functions, but on the other hand it seems reasonable to consider only those "innermost computations" that are also "computations".

We have:

$$\widetilde{\tau(F,\bar{b})}\ (\varphi,\ ) = \widetilde{\tau}(\varphi^+, <\widetilde{b}_1(\varphi,\ ),\ldots,\widetilde{b}_n(\varphi,\ ))$$ Lemma 3.3.2

$$= \widetilde{\tau}(\varphi^+, <b_1,b_2,\ldots,b_n>)$$ 2.3.2 a-ii

$$= \widetilde{\tau}(\varphi,\bar{b})$$ Lemma 2.3.5.1

$$= \varphi(\bar{b})$$ φ fixpoint of $\widetilde{\tau}$

$$= \widetilde{F(\bar{b})}\ (\varphi,\ )$$ 2.3.2 a-v .

Now, using Lemma 3.3.1, we obtain the desired property.

The Lemma follows by a trivial mathematical induction. □

Theorem 2 is now easily derived:

Theorem 2:

For every 'recursive definition', every weak fixpoint is an extension of every innermost computed function over D^n.

Proof: The proof is entirely analogous to that of Theorem 1, replacing '$\approx$' by '$\sim$', 'strong' by 'weak', 'Lemma 3.2.3' by 'Lemma 3.3.3', and 'computation' by 'innermost computation'. [Actually, we do not even have to use the fact that $\widetilde{\tau}(\varphi,\bar{\xi}) \in \Delta^+$, since the hypothesis of Lemma 3.3.3 is slightly weaker than that of Lemma 3.2.3].

□

The consequences of Theorem 2, and of combining Theorem 1 and 2 have been mentioned in Chapter 1 and we will not repeat them here, since the proofs follow trivially from the theorems.

CHAPTER 4

MONOTONICALLY STRUCTURED RECURSIVE DEFINITIONS

4.1 Introduction

4.2 Monotonicity. Continuity

4.2.1 General Definitions

4.2.2 Monotonic and Continuous Partial Functions

4.2.3 Monotonic Functionals

4.2.4 Continuous Functionals

4.3 Monotonically Structured Terms

4.3.1 Definition

4.3.2 Monotonicity of $\tilde{\tilde{\alpha}}[f]$ and $\tilde{\tilde{\alpha}}$

4.3.3 Monotonicity of $\tilde{\alpha}[f]$ and $\tilde{\alpha}$

4.3.4 Continuity of $\tilde{\tilde{\alpha}}[f]$ and $\tilde{\tilde{\alpha}}$

4.3.5 Continuity of $\tilde{\alpha}[f]$ and $\tilde{\alpha}$

4.4 Existence and Characterization of Least Fixpoints of Monotonically Structured Recursive Definitions

4.4.1 Least Fixpoints of Monotonic or Continuous Functionals

4.4.2 Least Strong Fixpoints (Theorem 3)

4.4.3 Least Weak Fixpoints (Theorem 4)

4.1 Introduction

In the previous chapters we have shown that the computed functions of recursive definitions were 'less than or equal to' the fixpoints of the recursive definitions, in the extension ordering, in a sense that has been made precise.

There are evidently situations where computed functions are strictly less defined than the fixpoints: a trivial illustration of this is provided by our previously made remark that the totally undefined function is a computed function of all but the constant recursive definitions.

Of course, what we are interested in is to compute least fixpoints (we know that the others cannot be computed functions). This is obviously not always possible, because some recursive definitions do not have least fixpoints, as has been shown in Chapter 1, (Section 1.6).

In this chapter, we give sufficient conditions on the recursive definitions which guarantee that they have least fixpoints of both types (weak and strong), and we give a characterization of these fixpoints.

4.2 Monotonicity. Continuity.

4.2.1 General Definitions

Let A,B be two partially ordered sets, and let us denote by $\prec$ the orderings on these sets. A mapping m of A into B is monotonic iff it preserves the orderings, i.e., iff:

$$\forall a,b \in A,\quad a \prec b \rightarrow m(a) \prec m(b).$$

Let A be a partially ordered set. A chain in A is a subset of A on which the ordering is total. A is chain-closed if every chain

in A has a least upper bound */ in A.

Let A and B be chain closed. A mapping m of A into B is continuous iff, for every chain K in A,

$$m(\text{lub}(K)) = \text{lub}(m(K)),$$

where this equation is intended to mean that m(K) must have a least upper bound in B, and that it must be equal to m(lub(K)).**/

Notice that continuity => monotonicity: Let m be a continuous mapping of A into B, and let $a,b \in A$, with $a \preccurlyeq b$. To show that $m(a) \preccurlyeq m(b)$ observe that $\{a,b\}$ is a chain in A, whose lub is b. By continuity, $\{m(a),m(b)\}$ must have m(b) as a lub in B. This implies $m(a) \preccurlyeq m(b)$.

The converse is not true in general, and we shall see examples of that in the sequel.

Finally, let A be a partially ordered set. An element ω in A is said to be a least element of A if $\forall a \in A, \omega \preccurlyeq a$.

Let us now use the previous definitions in the context of partial functions and functionals.

4.2.2 Monotonic and Continuous Partial Functions

We have seen various kinds of partial functions in the previous chapters:

Partial Functions of D^n into D	§2.3.2(a)	Function Variables
Partial Functions of $(D^+)^n$ into D	§2.3.2(b)	

*/ The notion of least upper bound of a subset of a partially ordered set is defined in Appendix I.

**/ If $S \subseteq A$ and m is a mapping of A into B, the notation m(S) represents the set $\{m(s) \mid s \in S\}$.

Partial Functions of D^n into Δ — §2.3.3(a): $\tilde{\tau}[f]$.

Partial Functions of $(D^+)^n$ into Δ — §2.3.3(b): $\tilde{\tilde{\tau}}[f]$.

Partial Functions of a Subset of $(\Delta^+)^p$ into Δ — §2.3.1 : Base functions.

All these can be considered as mappings of a set A into a set B, where A and B are subsets of $(\Delta^+)^p$ for some $p \geq 1$. We will define an ordering on such subsets in the following way (see Scott (1969)):

First define an ordering (denoted $\preceq$) on Δ^+ by:

$\forall x \in \Delta^+$, $\omega \preceq n$ and $x \preceq x$.

Then extend this ordering to $(\Delta^+)^p$, for any $p \geq 1$, by:

$\forall \bar{x}, \bar{y} \in (\Delta^+)^p$: $\bar{x} \preceq \bar{y}$ iff $(x_1 \preceq y_1)$ and $(x_2 \preceq y_2)$ and....and $(x_p \preceq y_p)$.

Finally also denote $\preceq$ the restriction of this ordering to any subset of $(\Delta^+)^p$.

Now the general definition of monotonicity given in §2.2.1 carries over immediately to the various kinds of partial functions.

First remark that $f \in pf_n(D)$ and $\tilde{\tau}[f] \in pf(D^n \to \Delta)$ are always monotonic: since D^n does not contain ω, the ordering on D^n is such that $\bar{x} \preceq \bar{y} \Rightarrow \bar{x} \equiv \bar{y}$.

Then: $f \in pf_n(D^+)$ is monotonic iff:

$\forall \bar{x}, \bar{y} \in (D^+)^n$, $\bar{x} \preceq \bar{y} \Rightarrow f(\bar{x}) \preceq f(\bar{y})$.

$\tilde{\tilde{\tau}}[f] \in pf((D^+)^n \to \Delta)$ is monotonic iff:

$\forall \bar{x}, \bar{y} \in (D^+)^n$, $\bar{x} \preceq \bar{y} \Rightarrow \tilde{\tau}[f](\bar{x}) \preceq \tilde{\tilde{\tau}}[f](\bar{y})$.

$g \in pf(Dom\ g \to \Delta)$ is monotonic iff:

$\forall \bar{x}, \bar{y} \in Dom\ g,\ \bar{x} \prec \bar{y} \Rightarrow g(\bar{x}) \prec g(\bar{y})$.

Note that, for these partial functions, <u>monotonicity implies continuity</u>. The key fact is that, in this case, A only has finite chains. (In $(\Delta^+)^k$, the longest chain has k+1 elements). Then, let:

$K = x_0 \prec x_1 \prec \ldots \prec x_p$, where $p \leq k$, be

a chain in A, and f be a monotonic mapping of A into B.

We have $f(lub(K)) = f(x_p)$.

But $f(K) = \{f(x_0), f(x_1), \ldots, f(x_p)\}$, and, by monotonicity

$f(x_0) \prec f(x_1) \prec \ldots \prec f(x_p)$. Hence $lub(f(K)) = f(x_p) = f(lub(K))$.

Let us now make some more remarks on monotonic partial functions, especially on the given functions. (The others are special cases of those, anyway).

As noted in Section 1, Chapter 1, any g such that:

$\forall \bar{x} \in Dom\ g\ [(\exists i,\ 1 \leq i \leq n)\ (x_i = \omega) \Rightarrow g(\bar{x}) = \omega]$

is monotonic, because in this case $\bar{x} \prec \bar{y}$ and $\bar{x} \neq \bar{y} \Rightarrow g(\bar{x}) = \omega$.

So any function g whose values are known for defined arguments can be extended on undefined arguments in such a way that it becomes monotonic. The easiest way of doing it is by setting $g(\bar{x}) = \omega$ whenever at least one of the x_i's is ω, which makes g monotonic as noted above. This is called the <u>natural extension</u> of g.

This is not the only way. For example, if g is a constant function whose domain is Δ^n (i.e., $\forall \bar{x} \in \Delta^n$, $g(\bar{x}) = a$, with $a \in \Delta$), then g' defined on $(\Delta^+)^n$ by: $\forall \bar{x} \in (\Delta^+)^n$ $g'(\bar{x}) = a$ is a monotonic extension of g which is not the natural extension.

In general, we may observe that for a function g to be monotonic,

it is necessary that whenever $g(\bar{y}) \equiv a \neq \omega$, for every $\bar{y} \succ \bar{x}$, $g(\bar{y}) \equiv a$. Hence if $\bar{x}$ has at least one undefined argument, for every $\bar{y}$ in the domain of g which is more defined than $\bar{x}$, $g(\bar{y})$ must be equal to $g(\bar{x})$. Using this, one may devise a way of monotonically extending g's which satisfy some equations. This can be stated more formally, but examples will convey the idea more clearly:

(a) Let us consider the binary multiplication '$*$' over the integers Z; this function satisfies the equations:

$$\forall x \in Z \quad 0 * x \equiv x * 0 \equiv 0 .$$

Hence we may extend '$*$' on $(Z^+)^2$ by:

$$0 * \omega \equiv \omega * 0 \equiv 0 .$$

This is a monotonic extension, which is not the natural extension. On all other combinations of arguments involving ω, '$*$' must take the value ω if it is to stay monotonic.

(b) The sequential 'if...then...else...' connective (See Paragraph 1.3.2) is another example of a monotonic, non-natural extension:

$$\forall x \in D: \text{ if } T \text{ then } x \text{ else } \omega = x$$

corresponds to the equation:

$$\forall x, y \in D, \text{ if } T \text{ then } x \text{ else } y = x .$$

(c) The parallel 'if...then...else...' connective (See Paragraph 1.3.2) is still a further monotonic extension, where:

$$\forall x \in D: \text{ if } \omega \text{ then } x \text{ else } x = x$$

corresponds to the equation:

$$\forall t \in \{T,F\}, \forall x \in D, \text{ if } t \text{ then } x \text{ else } x = x.$$

4.2.3 Monotonic Functionals

We have seen several kinds of functionals in previous chapters,

for example, in §2.3.3:

Functionals over $pf_n(D)$	$\tilde{\tau}$, for compatible τ
Functionals over $pf_n(D^+)$	$\tilde{\tilde{\tau}}$, for compatible τ
Functionals of $pf_n(D)$ into $pf(D^n \to \Delta)$	$\dot{\tilde{\tau}}$, for correct τ
Functionals of $pf_n(D^+)$ into $pf((D^+)^n \to \Delta)$	$\dot{\tilde{\tilde{\tau}}}$, for correct τ.

and we will see some more later in this chapter.

All these functionals are mappings of sets of partial functions into sets of partial functions. We know that the extension [*] relation $\leq$ is a partial ordering on those, and therefore the general definition of monotonicity carries over literally in this case.

4.2.4 Continuous Functionals

As in the previous case, the general definition of continuity carries over immediately in the case of functionals. Let A,B be chain closed sets of partial functions ordered by the extension relation $\leq$. Then a functional τ of A into B is continuous iff, for every chain K in A,

$$m(\text{lub}(K)) = \text{lub}(m(K)),$$

i.e., as in the general case, the right hand side is required to exist and be equal to the left hand side.

We will recall here a property of chains and their lubs in the case of partial functions, which will be of use later in this chapter. The proofs of the property appears in Appendix I.

Property: Every chain K in $pf(S \to R)$ has a lub, which satisfies the following:

---

[*] The extension relation has been defined in §2.2.3, footnote. See also Appendix I.

$\forall \xi \in S$: if $\forall f \in K$, $f(\xi) = \perp$ then $\mathrm{lub}(K)(\xi) = \perp$,

otherwise,

$\forall f \in K$, $f(\xi) \neq \perp$, $\mathrm{lub}(K)(\xi) = f(\xi)$.

If, in addition, every element in the chain is a monotonic partial function, then lub(K) is monotonic.

Hence $pf(S \to R)$ is chain closed, and so is $mf(S \to R)$, where $mf(S \to R)$ denotes the set of monotonic partial functions of $S \to R$.

Supposing the chain K is $\{f_i \mid i \geq 0\}$, with $i \leq j \Rightarrow f_i \leq f_j$, then the above property implies the following:

(P) $\forall \xi \in S$, there is $i_0 \geq 0$ such that, $\forall i \geq i_0$, $\mathrm{lub}(K)(\xi) = f_i(\xi)$

Notice that, in the case of functionals, monotonicity does not imply continuity. For example, denoting, Z the set of the integers, and $\perp$ the totally undefined function over Z, let us define the functional τ over $pf(Z)$ by:

$$\forall f \in pf(Z),\ \tau(f) = \begin{cases} f \text{ total} \to f \\ \text{otherwise} \to \perp \end{cases}$$

It is easy to see that τ is monotonic. However τ is not continuous: consider the chain $F = \{f_i \mid i \geq 0\}$,

$$\text{where } f_i(n) = \begin{cases} n \leq i \to 0 \\ \text{otherwise} \to \perp \end{cases}$$

For all $i \geq 0$ $\tau(f_i) = \perp$, so $\mathrm{lub}\,\tau(f_i) = \perp$ whereas $\mathrm{lub}(F)$ is the zero function 0, and $\tau(\mathrm{lub}\,F) = 0$.

4.3 Monotonically Structured Recursive Definitions

4.3.1 Definition

Let $\sigma(F,\bar{x})$ be a term over some interpreted alphabet. We say that σ is monotonically structured iff the interpretation g of every $\underset{\sim}{g}$

which appears in α is monotonic (as defined in 4.2.2). Notice that subterms of monotonically structured terms are monotonically structured.

A recursive definition $F(\bar{x}) \Leftarrow \tau(F,\bar{x})$ is said to be monotonically structured iff $\tau(F,\bar{x})$ is monotonically structured.

Monotonically structured recursive definitions are of special interest because they possess least fixpoints (Section 4.4) and these fixpoints are computable (Chapter 5).

We are now going to show monotonicity and continuity properties of the partial functions and functionals associated with monotonically structured terms.

4.3.2 Monotonicity of $\tilde{\alpha}[f]$ and α [*]

Lemma: Let α be a monotonically structured, correct term. Then, for every $f \in pf_n(D^+)$:

f monotonic $\Rightarrow$ $\tilde{\alpha}[f]$ monotonic.

Proof: Let $f \in pf_n(D^+)$ be monotonic, and let $\bar{\xi},\bar{\eta} \in (D^+)^n$ be such that $\bar{\xi} \sqsubseteq \bar{\eta}$.

We want to show $\tilde{\alpha}[f](\bar{\xi}) \sqsubseteq \tilde{\alpha}[f](\bar{\eta})$,

i.e. $\tilde{\alpha}(f,\bar{\xi}) \sqsubseteq \tilde{\alpha}(f,\bar{\eta})$, by 2.3.3(b).

We proceed by structural induction on α:

Case i: $\alpha = x_i$. Then $\tilde{\alpha}(f,\bar{\xi}) = \xi_i$, $\tilde{\alpha}(f,\bar{\eta}) = \eta_i$ and $\bar{\xi} \sqsubseteq \bar{\eta} \Rightarrow \xi_i \sqsubseteq \eta_i$.

Case ii: $\alpha = \underset{\sim}{c} \in C$. Then $\tilde{\alpha}(f,\bar{\xi}) = \tilde{\alpha}(f,\bar{\eta}) = c$.

Case iii: $\alpha = \underset{\sim}{\perp}$. Then $\tilde{\alpha}(f,\bar{\xi}) = \tilde{\alpha}(f,\bar{\eta}) = \perp$.

Case iv: $\alpha = \underset{\sim}{g}(\alpha_1,\ldots,\alpha_p)$.

Then $\tilde{\alpha}(f,\bar{\xi}) = g(\tilde{\alpha}_1(f,\bar{\xi}),\ldots,\tilde{\alpha}_p(f,\bar{\xi}))$

$\sqsubseteq g(\tilde{\alpha}_1(f,\bar{\eta}),\ldots,\tilde{\alpha}_p(f,\bar{\eta}))$ Induction hypothesis and monotonicity of g

---

[*]/For α correct, $\tilde{\alpha}[f]$ and α have been defined in 2.3.3(b).

$= \tilde{\tilde{\alpha}}(f,\bar{\eta})$.

Case (v): $\alpha = F(\alpha_1,\alpha_2,\ldots,\alpha_n)$.

Then $\tilde{\tilde{\alpha}}(f,\bar{\xi}) = f(\tilde{\tilde{\alpha}}_1(f,\bar{\xi}),\ldots,\tilde{\tilde{\alpha}}_n(f,\bar{\xi}))$

$\prec f(\tilde{\tilde{\alpha}}_1(f,\bar{\eta}),\ldots,\tilde{\tilde{\alpha}}_n(f,\bar{\eta}))$ (Induction hypothesis and monotonicity of f)

$= \tilde{\tilde{\alpha}}(f,\bar{\eta})$.

□

Let us call $mf_n(D^+)$ the set of all monotonic partial functions in $pf_n(D^+)$. i.e.: $mf_n(D^+) = \{f \in pf_n(D^+) \mid f \text{ is monotonic}\}$.

We have:

Lemma: Let α be a monotonically structured correct term. Then $\tilde{\tilde{\alpha}}$ is a monotonic functional of $mf_n(D^+)$ into $pf((D^+)^n \to \Delta)$.

Proof: This says:

$$\forall f,g \in mf_n(D^+): f \prec g \Rightarrow \tilde{\tilde{\alpha}}[f] \prec \tilde{\tilde{\alpha}}[g] ,$$

which is equivalent to:

$$\forall f,g \in mf_n(D^+): f \prec g \Rightarrow (\forall \bar{\xi} \in (D^+)^n)\ (\tilde{\tilde{\alpha}}(f,\bar{\xi}) \prec \tilde{\tilde{\alpha}}(h,\bar{\xi})) .$$

We prove the last property by structural induction on α.

Case (i): $\alpha = x_i$. Then $\tilde{\tilde{\alpha}}(f,\bar{\xi}) = \tilde{\tilde{\alpha}}(h,\bar{\xi}) = \xi_i$.

Case (ii): $\alpha = \underset{\sim}{c} \in C$. Then $\tilde{\tilde{\alpha}}(f,\bar{\xi}) = \tilde{\tilde{\alpha}}(h,\bar{\xi}) = c$.

Case (iii): $\alpha = \underset{\sim}{\omega}$. Then $\tilde{\tilde{\alpha}}(f,\bar{\xi}) = \tilde{\tilde{\alpha}}(h,\bar{\xi}) = \omega$.

Case (iv): $\alpha = g(\alpha_1,\ldots,\alpha_p)$.

Then, for all i, $1 \le i \le p$, $\tilde{\tilde{\alpha}}_i(f,\bar{\xi}) \prec \tilde{\tilde{\alpha}}_i(h,\bar{\xi})$ (Induction hypothesis).

Hence: $g(\tilde{\tilde{\alpha}}_1(f,\bar{\xi}),\ldots,\tilde{\tilde{\alpha}}_p(f,\bar{\xi})) \prec g(\tilde{\tilde{\alpha}}_1(h,\bar{\xi}),\ldots,\tilde{\tilde{\alpha}}_p(h,\bar{\xi}))$ (Monotonicity of g), i.e. $\tilde{\tilde{\alpha}}(f,\bar{\xi}) \prec \tilde{\tilde{\alpha}}(g,\bar{\xi})$.

Case (v): $\alpha = F(\alpha_1,\alpha_2,\ldots,\alpha_n)$.

Then, by induction hypothesis, $\forall i,\ 1 \le i \le n \quad \tilde{\tilde{\alpha}}_i(f,\bar{\xi}) \prec \tilde{\tilde{\alpha}}_i(g,\bar{\xi})$.

Hence: $\tilde{\alpha}(f,\bar{\xi}) = f(\tilde{\alpha}_1(f,\bar{\xi}),\ldots,\tilde{\alpha}_n(f,\bar{\xi}))$

$\leq f(\tilde{\alpha}_1(g,\bar{\xi}),\ldots,\tilde{\alpha}_n(g,\bar{\xi}))$ (Monotonicity of f)

$\leq g(\tilde{\alpha}_1(g,\bar{\xi}),\ldots,\tilde{\alpha}_n(g,\bar{\xi}))$ (Hypothesis $f \leq g$)

$= \alpha(g,\bar{\xi})$. □

If a term τ is compatible in addition to being monotonically structured, then $\tilde{\tau}$ is a monotonic functional over $mf_n(D^+)$:

Proof: if τ is compatible, and $f \in pf_n(D)$, then $\tilde{\tau}[f] \in pf_n(D)$ (§2.5.3). If $f \in mf_n(D^+)$, then $\tilde{\tau}f \in mf_n(D^+)$ by the first Lemma above, and $\tilde{\tau}$ is monotonic over $mf_n(D^+)$ by the second Lemma above. □

Notice that there are terms τ which are not monotonically structured but which are such that $\tilde{\tau}$ is monotonic. One example is the following:

$$\tau(F,x) = \underset{\sim}{\text{if}}\ F(x) = 0\ \underset{\sim}{\text{then}}\ F(x)\ \underset{\sim}{\text{else}}\ 0.$$

It is not monotonically structured because $=$ is not monotonic, but $\tilde{\tau}$ is monotonic, because it maps every $f \in pf(D^+)$ into the zero function.

Notice that in this case, $\tau(F,x)$ can be replaced by $\tau'(F,x)$ in such a way that $\tilde{\tau}' = \tilde{\tau}$, with τ' monotonically structured, the most simple τ' being 0. For another one, see Chapter 1, Section 1.[illegible], Example 5).

We don't know whether this is possible in general.

Notice also, that the last lemma above is false if the function variable can take non-monotonic values in $pf_n(D^+)$. For example, if we take $\tau(F,x) = F(F(x))$ and if we consider $f \in pf(D^+)$: $\begin{cases} x \to 0 \\ \text{otherwise} \to x \end{cases}$,

and $h \in pf(D^+)$: $\begin{cases} x \to 0 \\ \text{otherwise} \to 1, \end{cases}$

we have $f \prec h$, but not $\tilde{\tilde{\tau}}[f] \prec \tilde{\tilde{\tau}}[h]$, since $\tilde{\tilde{\tau}}[f](0) \equiv 0$ and $\tilde{\tilde{\tau}}[h](0) \equiv 1$. So, even though τ is monotonically structured, $\tilde{\tilde{\tau}}$ is not monotonic over $pf(D^+)$. □

4.3.3 Monotonicity of $\tilde{\alpha}[f]$ and $\tilde{\alpha}$ */

We will briefly mention here the monotonicity properties for $\tilde{\alpha}[f]$ and $\tilde{\alpha}$.

First, notice that every $f \in pf_n(D)$ is trivially monotonic, as observed in §4.2.2 above.

Similarly, for every $f \in pf_n(D)$, $\tilde{\alpha}[f]$ is monotonic. (Trivial, by the same observation.)

The analogous of the second lemma of §4.3.2 is:

Lemma. Let α be a monotonically structured, correct term. Then $\tilde{\alpha}$ is a monotonic functional.

Proof: $\tilde{\alpha}$ is a monotonic functional iff:

for every $f, h \in pf_n(D)$:

$$f \prec h \Rightarrow (\forall \bar{\xi} \in D^n)\ (\tilde{\alpha}(f,\bar{\xi}) \prec \tilde{\alpha}(h,\bar{\xi})).$$

Let $f,h \in pf_n(D)$ such that $f \prec h$, and $\bar{\xi} \in D^n$. We have:

$\tilde{\alpha}(f,\bar{\xi}) = \tilde{\tilde{\alpha}}(f^+,\bar{\xi})$ (Paragraph 2.3.5.1).

Now f^+ and h^+ are trivially monotonic, and since $f \prec h$ we have $f^+ \prec h^+$. Furthermore, $D^n \subseteq (D^+)^n$, and we can apply the second lemma of previous section to get:

$$\tilde{\tilde{\alpha}}(f^+,\bar{\xi}) \prec \tilde{\tilde{\alpha}}(h^+,\bar{\xi}).$$

Hence, using transitivity and Paragraph 2.3.5.1 once more:

$$\tilde{\alpha}(f,\bar{\xi}) \prec \tilde{\alpha}(h,\bar{\xi})$$ □

If, in addition to being monotonically structured, the term τ is compatible, then $\tilde{\tau}$ is a monotonic functional over $pf_n(D)$.

*/ For α correct, $\tilde{\alpha}[f]$ and $\tilde{\alpha}$ have been defined in §2.3.3(a).

4.3.4. Continuity of $\tilde{\tilde{\alpha}}[f]$ and $\tilde{\tilde{\alpha}}$

Since $\tilde{\tilde{\alpha}}[f]$ is a partial function, monotonicity is equivalent to continuity, and the first Lemma of 4.3.3 tells us that, for every $f \in pf_n(D^+)$:

f monotonic $\Rightarrow \tilde{\tilde{\alpha}}[f]$ continuous.

The continuity property of $\tilde{\tilde{\alpha}}$, which corresponds to the second Lemma of 4.3.3 and is actually strictly stronger, is:

Lemma: Let α be a monotonically structured correct term. Then $\tilde{\tilde{\alpha}}$ is a continuous functional of $mf_n(D^+)$ into $mf((D^+)^n \to \Delta)$.

Proof: We already know from 4.3.3 that $\tilde{\tilde{\alpha}}$ is a monotonic functional of $mf_n(D^+)$ into $mf((D^+)^n \to \Delta)$.

Let K be a chain of $mf_n(D^+)$. Then we know that $\tilde{\tilde{\alpha}}(K)$ is a chain of $mf((D^+)^n \to \Delta)$, and therefore that $\mathrm{lub}\,\tilde{\tilde{\alpha}}(K)$ exists and belongs to the same set.*/

To prove continuity, there remains to show that:

$$\tilde{\tilde{\alpha}}[\mathrm{lub}(K)] = \mathrm{lub}\,\tilde{\tilde{\alpha}}(K).$$

We will do this by structural induction on α.

Case (i): $\alpha = x_j$, $1 \le j \le n$.

Then, for every $f \in pf_n(D^+)$, for every $\vec{\xi} \in (D^+)^n$, we have:

$\tilde{\tilde{\alpha}}[f]\vec{\xi} = f(\vec{\xi}) = \xi_j$.

In particular, $\tilde{\tilde{\alpha}}[\mathrm{lub}(K)](\vec{\xi}) = \xi_j$.

On the other hand, we also have $\forall i$, $\tilde{\tilde{\alpha}}[f_i](\vec{\xi}) = \xi_j$,

so, by definition of lub : $\mathrm{lub}(\tilde{\tilde{\alpha}}(K))(\vec{\xi}) = \xi_j$.

Hence: $\tilde{\tilde{\alpha}}[\mathrm{lub}(K)] = \mathrm{lub}\,\tilde{\tilde{\alpha}}(K)$.

Case (ii): $\alpha = \underset{\sim}{c} \in C$.

Then, for every $f \in pf_n(D^+)$, for every $\vec{\xi} \in (D^+)^n$,

$\tilde{\tilde{\alpha}}[f](\vec{\xi}) = c$.

*/ Actually, one could prove continuity without using the fact that $\tilde{\tilde{\alpha}}$ is monotonic over $mf_n(D^+)$, and then indeed deduce directly monotonicity of $\tilde{\tilde{\alpha}}$, since continuity $\Rightarrow$ monotonicity.

So $\tilde{\tilde{\alpha}}[\mathrm{lub}(K)](\bar{\xi}) \equiv c$.

On the other hand, we also have, $\forall i$, $\tilde{\tilde{\alpha}}[f_i](\bar{\xi}) \equiv c$, so $\mathrm{lub}(\tilde{\tilde{\alpha}}[K])(\bar{\xi}) \equiv c$.

Hence: $\tilde{\tilde{\alpha}}[\mathrm{lub}(K)] = \mathrm{lub}(\tilde{\tilde{\alpha}}[K])$

Case (iii): $\alpha = \underset{\sim}{w}$ Analogous to case (ii).

Case (iv): $\alpha = \underset{\sim}{g}(\alpha_1, \alpha_2, \ldots, \alpha_p)$.

Let $\bar{\xi} \in (D^+)^n$.

We have: $\tilde{\tilde{\alpha}}[\mathrm{lub}(K)](\bar{\xi}) \equiv g(\tilde{\tilde{\alpha}}_1[\mathrm{lub}(K)](\bar{\xi}), \ldots, \tilde{\tilde{\alpha}}_p[\mathrm{lub}(K)](\bar{\xi}))$

$\equiv g(\mathrm{lub}(\tilde{\tilde{\alpha}}_1[K])(\bar{\xi}), \ldots, \mathrm{lub}(\tilde{\tilde{\alpha}}_p[K](\bar{\xi}))$,

by induction hypothesis,

$\equiv g(a_1, a_2, \ldots, a_p)$,

where we designate $\mathrm{lub}(\tilde{\tilde{\alpha}}_j[K])(\bar{\xi})$ by a_j, for every j, $1 \leq j \leq p$.

For every $f_i \in K$, $\tilde{\tilde{\alpha}}[f_i](\bar{\xi}) \equiv g(\tilde{\tilde{\alpha}}_1[f_i](\bar{\xi}), \ldots, \tilde{\tilde{\alpha}}_p[f_i](\bar{\xi}))$

$\equiv g(a_1^i, \ldots, a_p^i)$,

$\equiv a_i$,

where we have designated $\tilde{\tilde{\alpha}}_j[f_i](\bar{\xi})$ by a_j^i, for every j, $1 \leq j \leq p$, and $g(a_1^i, \ldots, a_p^i)$ by a_i.

Now, we know that $\tilde{\tilde{\alpha}}[K]$ is a chain, and so, because of that, there is an i, say i_0, such that:

$\forall i \geq i_0$, $a_i \equiv a_{i_0} \equiv \mathrm{lub}(\tilde{\tilde{\alpha}}[K])(\bar{\xi})$. (Property P, Paragraph 4.2.4.)

Also, for every j, $1 \leq j \leq p$, $\tilde{\tilde{\alpha}}_j[K]$ is a chain, and so there is an i, say i_j, such that $\forall i \geq i_j$, $a_j^i \equiv a_j^{i_j} \equiv \mathrm{lub}(\tilde{\tilde{\alpha}}_j[K])(\bar{\xi}) \equiv a_j$ (Property P, Paragraph 4.2.4).

Now, for any $i \geq \max(i_0, i_1, \ldots, i_p)$, we have:

$$\text{lub}(\tilde{\alpha}[K])(\bar{\xi}) \equiv a_i \quad (\text{since } i \geq i_o)$$

$$\equiv g(a_1^i,\ldots,a_p^i) \quad (\text{Definition of } a_i)$$

$$\equiv g(a_1,\ldots,a_p) \quad (\text{Since } i \geq i_j, \text{ for every } j,\ 1 \leq j \leq p)$$

$$\equiv \tilde{\tilde{\alpha}}[\text{lub}(K)](\bar{\xi}).$$

Hence $\text{lub}(\tilde{\alpha}[K]) = \tilde{\tilde{\alpha}}[\text{lub}(K)]$.

Case (v): $\alpha = F(\alpha_1,\ldots,\alpha_n)$.

Let $\bar{\xi} \in (D^+)^n$.

We have: $\tilde{\alpha}[\text{lub}(K)](\bar{\xi}) \equiv \text{lub}(K)\ (\tilde{\tilde{\alpha}}_1[\text{lub}(K)](\bar{\xi}),\ldots,\tilde{\tilde{\alpha}}_n[\text{lub}(K)](\bar{\xi}))$

$$\equiv \text{lub}(K)(\text{lub}(\tilde{\tilde{\alpha}}_1[K])(\bar{\xi}),\ldots,\text{lub}(\tilde{\tilde{\alpha}}_n[K])(\bar{\xi})),$$

by induction hypothesis.

$$\equiv \text{lub}(K)(a_1,a_2,\ldots,a_n),$$

where we again designate $\text{lub}(\tilde{\tilde{\alpha}}_j[K])(\bar{\xi})$ by a_j, for every j, $1 \leq j \leq n$.

Now for every $f_i \in K$, we have:

$$\tilde{\tilde{\alpha}}[f_i](\bar{\xi}) \equiv f_i(\tilde{\tilde{\alpha}}_1[f_i](\bar{\xi}),\ldots,\tilde{\tilde{\alpha}}_n[f_i](\bar{\xi}))$$

$$\equiv f_i(a_1^i,\ldots,a_n^i)$$

$$\equiv a_i,$$

where we again designate $\tilde{\tilde{\alpha}}_j[f_i](\bar{\xi})$ by a_j^i, for every j, $1 \leq j \leq n$, and $f_i(a_1^i,\ldots,a_n^i)$ by a_i. Now, we know that $\tilde{\alpha}[K]$ is a chain, and so, because of that, there is an i, say i_o, such that:

$$\forall\ i \geq i_o,\ a_i \equiv a_{i_o} \equiv \text{lub}(\tilde{\tilde{\alpha}}[K])(\bar{\xi}). \quad \text{(Property P, Paragraph 4.2.4).}$$

Also, for every j, $1 \leq j \leq n$, $\tilde{\tilde{\alpha}}_j[K]$ is a chain, and there is an i, say i_j such that:

$$\forall\ i \leq i_j,\ a_j^i \equiv a_j^{i_j} \equiv \text{lub}(\tilde{\tilde{\alpha}}_j[K])(\bar{\xi}) \equiv a_j$$

(Property P, Paragraph 4.2.4).

Finally, since K is a chain, there must be an i, say i_M, such that:

$$\forall\, i \geq i_M,\ f_i(a_1,a_2,\ldots,a_n) = \mathrm{lub}(K)(a_1,a_2,\ldots,a_n)$$

where $a_1,a_2,\ldots,a_n$ have been defined above. (Property P).

Now, for any $i \geq \max(i_o,i_1,i_2,\ldots,i_n,i_M)$, we have:

$$\begin{aligned}
\mathrm{lub}(\tilde{\tilde{\alpha}}[K])(\bar{\xi}) &= a_i && \text{(Since } i \geq i_o\text{)}\\
&= f_i(a_1^i,\ldots,a_n^i) && \text{(Def. of } a_i\text{)}\\
&= f_i(a_1,\ldots,a_n) && \text{(Since } i \geq i_j \text{ for every } i,\ 1 \leq j \leq n\text{)}\\
&= \mathrm{lub}(K)(a_1,\ldots,a_n) && \text{(Since } i \geq i_M\text{)}\\
&= \tilde{\alpha}[\mathrm{lub}(K)](\bar{\xi}) && \text{(First derivation of Case (v)).}
\end{aligned}$$

This completes the proof that $\mathrm{lub}(\tilde{\tilde{\alpha}}[K]) = \tilde{\tilde{\alpha}}[\mathrm{lub}(K)]$. □

4.3.5 Continuity of $\tilde{\alpha}[\,]$ and $\tilde{\alpha}$

$\tilde{\alpha}[f]$ is trivially continuous, since it is a monotonic partial function.

At the functional level, we have:

Lemma: Let α be a monotonically structured, correct term. Then $\tilde{\alpha}$ is a continuous functional.

Proof: Let K be a chain in $pf_n(D)$, and denote $K^+ = \{f^+ \mid f \in K\}$. Then K^+ is a chain in $mf_n(D^+)$, and so, by §4.3.4, we have:

$$\tilde{\tilde{\alpha}}[\mathrm{lub}(K^+)] = \mathrm{lub}(\tilde{\tilde{\alpha}}[K^+]).$$

i.e. $$\forall\, \bar{\xi} \in D^n,\ \tilde{\tilde{\alpha}}[\mathrm{lub}(K^+)](\bar{\xi}) = \mathrm{lub}(\tilde{\tilde{\alpha}}[K^+])(\bar{\xi}).$$

Now, we observe that $\mathrm{lub}(K^+) = (\mathrm{lub}\ K)^+$, and that $\tilde{\tilde{\alpha}}[K^+] = \tilde{\alpha}[K]$ on D^n; therefore we have:

$$\forall\, \bar{\xi} \in D^n,\ \tilde{\tilde{\alpha}}[(\mathrm{lub}\ K)^+](\bar{\xi}) = \mathrm{lub}(\tilde{\alpha}[K])(\bar{\xi}),$$

i.e. $$\forall\, \bar{\xi} \in D^n,\ \tilde{\alpha}[\mathrm{lub}\ K](\bar{\xi}) = \mathrm{lub}(\tilde{\alpha}[K])(\bar{\xi}),$$

or: $$\tilde{\alpha}[\mathrm{lub}\ K] = \mathrm{lub}(\tilde{\alpha}[K]),$$

which expresses the continuity of α.

□

4.4 Existence and Characterization of the Least Fixpoints of Monotonically Structured Recursive Definitions

4.4.1 Least Fixpoints of Monotonic or Continuous Functionals

Let M be a set, and m a mapping of M into M. A fixpoint of m is an element s of M such that $s = m(s)$.

Let $(M, \leq)$ be a partially ordered set, and m a mapping of M into M. A least fixpoint of m is a fixpoint s of m such that, for every fixpoint t of m, $s \leq t$.

The following general theorems are proved in Appendix I:

Theorem: If M is a partially ordered set which is chain closed and has a least element, every monotonic mapping m of M into itself has a least fixpoint in M.

Theorem: If M is a partially ordered set which is chain closed and has a least element Ω, and if m is a continuous mapping m of M into itself, then the least fixpoint of m is: $\text{lub}\{m^i(\Omega) \mid i \geq 0\}$.

(The notation $m^i(a)$, where $a \in M$, is defined inductively by:

$$m^0(a) = a; \quad \forall i \geq 0, m^{i+1}(a) = m(m^i(a))).$$

These theorems will enable us to show the existence and give a characterization of the least fixpoints of monotonically structured recursive definitions.

4.4.2 Least Strong Fixpoint

Theorem 3: Every monotonically structured recursive definition $F(\bar{x}) \Leftarrow \tau(F,\bar{x})$ has a monotonic least strong fixpoint, $\hat{\hat{f}}_\tau$. In addition:

$$\hat{\hat{f}}_\tau = \text{lub}\{\tilde{\tilde{\tau}}^i(\Omega) \mid i \geq 0\}$$

Proof: By definition, a strong fixpoint of the recursive definition is a fixpoint of $\tilde{\tilde{\tau}}$. Since τ is monotonically structured and

compatible, we know that $\tilde{\tau}$ is monotonic over $mf_n(D^+)$ by §4.3.2, and even continuous over $mf_n(D^+)$, by §4.3.4.

Now the undefined function Ω of $pf_n(D^+)$ is monotonic, and therefore $mf_n(D^+)$ has a least element. We also know, by §4.2.4, that $mf_n(D^+)$ is chain closed, and therefore we can apply both theorems of the previous paragraph with $M = mf_n(D^+)$ and $m = \tilde{\tau}$. This immediately yields Theorem 3. □

4.4.3 Least Weak Fixpoint.

Theorem 4. Every monotonically structured recursive definition $F(\bar{x}) \Leftarrow \tau(F,\bar{x})$ has a monotonic least weak fixpoint, $\hat{f}_\tau$. In addition:

$$\hat{f}_\tau = \text{lub}\{\tilde{\tau}^i(\Omega) \mid i \geq 0\} .$$

Proof: Entirely analogous to that of Theorem 3. By definition, a weak fixpoint of the recursive definition is a fixpoint of $\tilde{\tau}$. Since τ is monotonically structured and compatible, we know that $\tilde{\tau}$ is monotonic over $pf_n(D)$ by §4.3.3 and even continuous over $pf_n(D)$ by §4.3.5.

Now $pf_n(D)$ has a least element, the totally undefined function Ω, and is chain closed (§4.2.4), and therefore we can apply both Theorems of Paragraph 4.4.1 with $M = pf_n(D)$ and $m = \tilde{\tau}$. This immediately yields Theorem 4.

It will be shown later (Comment 2, at the end of Paragraph 5.4.3) that $\hat{f}_\tau$ extends $\hat{f}_\tau$ in the sense that $\hat{f}^+_\tau \leq \hat{f}_\tau$ (where f^+ denotes the natural extension of f, as defined in Paragraph 2.3.5).

CHAPTER 5

FIXPOINT COMPUTATIONS

5.1 Introduction

5.2 Standard Simplifications

5.2.1 Standard Simplification Schemas and Rules

5.2.2 Standard Simplification Relation

5.2.3 Standard Simplifications

5.3 Full Computations

5.3.1 Full Substitutions

5.3.2 Full Computations

5.3.3 Theorem 5

5.4 Standard Innermost Computations

5.4.1 Parallel Innermost Substitutions

5.4.2 Standard Innermost Computations

5.4.3 Theorem 6

5.5 Safe Innermost Computations

5.5.1 Safe Simplifications

5.5.2 Safe Innermost Substitutions

5.5.3 Safe Innermost Computations

5.5.4 Theorem 7

5.1 Introduction

In this chapter, we essentially provide computation rules which guarantee that the corresponding computed function of a monotonically structured recursive definition is one of the fixpoints of which we have proven the existence in the previous Chapter.

We first define standard simplifications and full substitutions, and give the 'full computation rule', which leads to the least strong fixpoint.

Then we describe the 'standard innermost computation rule', which leads to the least weak fixpoint.

We finally give a large class of computation rules, which we call 'safe innermost computation rules', and which also lead to the least weak fixpoint.

5.2 Standard Simplifications

5.2.1 Standard Simplification Schemas and Rules

Definition: A standard simplification schema is any expression of the form:

$$\underset{\sim}{g}(A_1, A_2, \ldots, A_n) \rightarrow \underset{\sim}{a} \ ,$$

where:

(i) each A_i is an individual constant or a term variable (a term variable is a letter which stands for an arbitrary term);

(ii) no two term variables are identical;

(iii) $\underset{\sim}{a}$ is an individual constant in C;

(iv) the equation $g(\xi_1, \xi_2, \ldots, \xi_r) = a$ holds for all values of $(\xi_1, \xi_2, \ldots, \xi_n)$ in $Dom(g)$ such that, if A_j is an individual constant $\underset{\sim}{a}_i$, then ξ_i is its value a_i.

Intuitively, this says that a standard simplification schema for some given function $\underset{\sim}{g}$ corresponds to the property that specifying some

of the arguments of g to be constant causes g to be a constant, no matter what is the value of the other arguments.

A standard simplification rule is any instance of a standard simplification schema where all the term variables have been replaced by arbitrary terms in such a way that the resulting left hand side term is correct.

For example:

(a) $\underset{\sim}{\text{if}}\ \underset{\sim}{\text{T}}\ \underset{\sim}{\text{then}}\ \underset{\sim}{\text{a}}\ \underset{\sim}{\text{else}}\ \underset{\sim}{\text{A}} \rightarrow \underset{\sim}{\text{a}}$ is a standard simplification schema for the sequential 'if -- then -- else', corresponding to the equation: $\forall\, x \in D^{+}$: if T then a else x $\equiv$ a.

(b) $\underset{\sim}{\text{if}}\ \underset{\sim}{\text{T}}\ \underset{\sim}{\text{then}}\ \underset{\sim}{\text{a}}\ \underset{\sim}{\text{else}}\ \underset{\sim}{\text{F}}(x) \rightarrow \underset{\sim}{\text{a}}$ is a standard simplification rule for the sequential 'if -- then -- else' connective which is an instance of the above schema.

(c) $\underset{\sim}{\text{if}}\ \underset{\sim}{\text{T}}\ \underset{\sim}{\text{then}}\ \underset{\sim}{\text{A}}\ \underset{\sim}{\text{else}}\ \underset{\sim}{\text{B}} \rightarrow \underset{\sim}{\text{A}}$ is not a standard simplification schema for the sequential 'if -- then -- else' connective because A is not a constant.

(d) $\underset{\sim}{\text{if}}\ \underset{\sim}{\text{A}}\ \underset{\sim}{\text{then}}\ \underset{\sim}{\text{B}}\ \underset{\sim}{\text{else}}\ \underset{\sim}{\text{B}} \rightarrow \underset{\sim}{\text{B}}$ is not a standard simplification schema for the parallel 'if -- then -- else' connective, because the term variable B appears twice in the left hand side expression.

(e) for any given function g, if $\langle a_1, a_2, \ldots, a_p \rangle \in \text{Dom}(g)$ and $g(a_1, a_2, \ldots, a_p) = a$, with $a \neq \omega$, then $\underset{\sim}{g}(\underset{\sim}{a}_1, \underset{\sim}{a}_2, \ldots, \underset{\sim}{a}_p) \rightarrow \underset{\sim}{a}$ is a standard simplification schema (or rule).

(f) Notice that expressions of the form: $\underset{\sim}{g}(\underset{\sim}{A}_1, \underset{\sim}{A}_2, \ldots, \underset{\sim}{A}_n) \rightarrow \underset{\sim}{\omega}$ are <u>not</u> standard simplification schemas, because, in the definition of such, the righthand side must belong

to C, hence cannot be $\underset{\sim}{\omega}$.

(g) If '$*$' is the ordinary binary multiplication over the integers extended by $0 * \omega = \omega$ [Natural extension], then $\underset{\sim}{0} * A \to \underset{\sim}{0}$ is not a standard simplification schema (because the equation $0 * \xi = 0$ does not hold for $\xi = \omega$, although $(0,\omega) \in \mathrm{Dom}(*)$).

However, if '$*$' is the ordinary multiplication over the integers now extended by $0 * \omega = 0$ (which is a monotonic extension), then $\underset{\sim}{0} * A \to \underset{\sim}{0}$ is a standard simplification schema.

At this point, let us emphasize that there are a number of simplification schemas that one might want to consider other than the standard ones, but the fact is that the standard ones are sufficient to obtain fixpoint computations, as will be seen later.

Standard simplification rules have the obvious following properties:

(1) Let $\beta \to \gamma$ be a standard simplification rule. Then:

$$\forall f \in pf_n(D^+),\ \forall \overline{\xi} \in (D^+)^n,\ \tilde{\beta}(f,\overline{\xi}) = \tilde{\gamma}(f,\overline{\xi}).$$

Proof: Let $f \in pf_n(D^+)$, $\overline{\xi} \in (D^+)^n$. Then we have:

$\beta = \underset{\sim}{g}(\beta_1,\beta_2,\ldots,\beta_p)$, $\gamma = \underset{\sim}{a}$ for some $g \in G_p$ and some $\underset{\sim}{a} \in C$. If we denote $\tilde{\beta}_i(f,\overline{\xi})$ by b_i for $1 \le i \le p$, we know that $(b_1,b_2,\ldots,b_p) \in \mathrm{Dom}(g)$, because β is a correct term. And, because of the definition of a standard simplification rule, we have:

$$g(b_1,b_2,\ldots,b_n) = a,$$

which implies: $\tilde{\beta}(f,\overline{\xi}) = \tilde{\gamma}(f,\overline{\xi})$. □

(2) Also: $\forall f \in pf_n(D),\ \forall \overline{x} \in D^n,\ \tilde{\beta}(f,\overline{x}) = \tilde{\gamma}(f,\overline{x})$.

Proof: trivial by the above property and Lemma 2.3.5.1. □

5.2.2 Standard Simplification Relation

Definition: The standard simplification relation is a relation between terms. It consists of the set of pairs of terms $\langle \alpha, \beta \rangle$ such that:

(a) $\beta \neq \alpha$;

(b) $\beta \in S^1_{\gamma,\delta}\alpha$, where $\gamma \to \delta$ is a standard simplification rule.

This relation will be denoted $\overset{S}{\to}$.

Remarks: (1) If $\alpha \overset{S}{\to} \beta$, then, $\forall f \in pf_n(D^+)$, $\forall \bar{\xi} \in (D^+)^n$,

$$\forall(f,\bar{\xi}) = \mathcal{D}(f,\bar{\xi}).$$

This is trivial from the observation at the end of the previous paragraph and Lemma 4.2.1 of Chapter 3.

(2) If $\alpha \overset{S}{\to} \beta$ and α is correct, β is correct.

(Trivial from previous remark).

(3) If $\alpha \overset{S}{\to} \beta$, then $|\beta| < |\alpha|$, where, for any term τ, $|\tau|$ denotes the size of τ (number of symbols in τ).*/

Proof: the proof is immediate. It comes from the fact that, in a standard simplification rule $\underset{\sim}{g}(\tau_1,\ldots,\tau_p) = \underset{\sim}{a}$, the length of the right hand side is strictly smaller than the length of the left hand side, and from the fact that we have excluded $\beta = \alpha$ when $\alpha \overset{S}{\to} \beta$. A rigorous induction argument can easily be built using these two observations. □

We denote by $\left(\overset{S}{\to}\right)^*$ the transitive reflexive closure of $\overset{S}{\to}$. That is: $\alpha\left(\overset{S}{\to}\right)^*\beta$ if and only if $\alpha = \beta$ or there is a positive integer N and a sequence $\{\alpha_i \mid 0 \leq i \leq N\}$ such that:

(a) $\alpha_0 = \alpha$;

*/ The size $|\tau|$ of a term τ can be defined inductively in the obvious way: $|x_i| = 1$ for every i, $1 \leq i \leq n$; $|\underset{\sim}{c}| = 1$ for every $\underset{\sim}{c} \in C$; $|\omega| = 1$; $|\underset{\sim}{g}(\tau_1,\ldots,\tau_p)| = |\tau_1|+\cdots+|\tau_p|+1$; $|F_k(\tau_1,\tau_1,\ldots,\tau_n)| = |\tau_1|+\cdots+|\tau_n|+1$.

(b) for every i, $0 \leq i < N$, $\alpha_i \xrightarrow{s} \alpha_{i+1}$;

(c) $\alpha_N = \beta$.

5.2.3 Standard Simplifications

Let α be a term free of $\bar{x}$. A standard simplification of α is a sequence of terms $\alpha_0 \xrightarrow{s} \alpha_1 \xrightarrow{s} \alpha_2 \xrightarrow{s} \ldots$

such that:

(a) $\alpha_0 = \alpha$;

(b) if $i > 0$ and α_i is in the sequence, then: $\alpha_{i-1} \xrightarrow{s} \alpha_i$;

(c) if α_n has no successor in the sequence, then there is no β such that $\alpha_n \xrightarrow{s} \beta$.

Notice that, because of the remark 2 above, a standard simplification is necessarily a finite sequence. If α_n is the last term of the sequence, the simplification is said to be of length n. A comparison with the definition of computations in 2.4.2 shows that a standard simplification of α is in fact a finite initial subsequence of a computation of α.

We are now going to prove an interesting property of the relation $\xrightarrow{s}$:

Lemma: For every term α, if $\alpha \xrightarrow{s} \alpha_1$ and $\alpha \xrightarrow{s} \alpha_2$, then there is a term γ such that $\alpha_1 (\xrightarrow{s})^* \gamma$ and $\alpha_2 (\xrightarrow{s})^* \gamma$.

Proof: By structural induction on α:

Cases (i), (ii), (iii): if $\alpha = x_i$, or $\alpha = \underset{\sim}{c} \in C$, or $\alpha = \underset{\sim}{\omega}$, then the result is vacuously true, since there can be no β such that $\alpha \xrightarrow{s} \beta$ by remark 2 above.

Case iv: $\alpha = \underset{\sim}{g}(\tau_1, \tau_2, \ldots, \tau_p)$: this case can be subdivided into 4 subcases:

iv-1: $\alpha_1 = \underset{\sim}{g}(\tau_1, \ldots, \tau_{i-1}, \tau_i', \tau_{i+1}, \ldots, \tau_p)$,

where $\tau_i \xrightarrow{s} \tau_i'$,

and: $\alpha_2 = \underset{\sim}{g}(\tau_1,\ldots,\tau_{j-1},\tau_j',\tau_{j+1},\ldots,\tau_p)$,

where $\tau_j \xrightarrow{s} \tau_j'$, and $i \neq j$ (suppose $i < j$, for example).

Then we can take $\gamma = \underset{\sim}{g}(\tau_1,\ldots,\tau_{i-1},\tau_i',\tau_{i+1},\ldots,\tau_{j-1},\tau_j',\tau_{j+1},\ldots,\tau_p)$ since we can apply $\tau_j \xrightarrow{s} \tau_j'$ in α_1 to get $\alpha_1 \xrightarrow{s} \gamma$, and similarly $\tau_i \xrightarrow{s} \tau_i'$ in α_2 will give $\alpha_2 \xrightarrow{s} \gamma$.

<u>iv-2</u>: $\alpha_1 = \underset{\sim}{g}(\tau_1,\ldots,\tau_i',\ldots,\tau_p)$ where $\tau_i \xrightarrow{s} \tau_i'$,

and: $\alpha_2 = \underset{\sim}{g}(\tau_1,\ldots,\tau_i'',\ldots,\tau_p)$ where $\tau_i \xrightarrow{s} \tau_i''$.

In this case, by the induction hypothesis on τ_i, we know that there is a term γ_i such that $\tau_i' \overset{s}{\rightarrow}{}^{*} \gamma_i$ and $\tau_i'' \overset{s}{\rightarrow}{}^{*} \gamma_i$. Therefore $\alpha_1 \overset{s}{\rightarrow}{}^{*} \gamma$ and $\alpha_2 \overset{s}{\rightarrow}{}^{*} \gamma$, where:

$$\gamma = \underset{\sim}{g}(\tau_1,\ldots,\gamma_i,\ldots,\tau_p) \quad .$$

<u>iv-3</u>: $\alpha_1 = \underset{\sim}{g}(\tau_1,\ldots,\tau_i',\ldots,\tau_p)$, where $\tau_i \xrightarrow{s} \tau_i'$,

and there is a standard simplification rule :

$\alpha = \underset{\sim}{g}(\tau_1,\ldots,\tau_p) \rightarrow \alpha_2 = \underset{\sim}{a}$, with $\underset{\sim}{a} \in C$.

Let $\underset{\sim}{g}(A_1,\ldots,A_i,\ldots A_p) \rightarrow \underset{\sim}{a}$ be a standard simplification schema of which $\underset{\sim}{g}(\tau_1,\ldots,\tau_i,\ldots,\tau_p) \rightarrow \underset{\sim}{a}$ is an instance. Notice that A_i is not a constant, since τ_i is an instance of A_i and we have assumed the existance of τ_i' such that $\tau_i \xrightarrow{s} \tau_i'$. Then $\underset{\sim}{g}(\tau_1,\ldots,\tau_i',\ldots,\tau_p) \rightarrow \underset{\sim}{a}$ is another instance of the same schema, where A_i is replaced by the term τ_i' and all other A_j, $j \neq i$, are replaced by τ_j. Therefore $\underset{\sim}{g}(\tau_1,\ldots,\tau_i',\ldots,\tau_p) \rightarrow \underset{\sim}{a}$ is a standard simplification rule, and $\alpha_1 \xrightarrow{s} \underset{\sim}{a} = \alpha_2$. Therefore $\gamma = \alpha_2$ will be suitable.

iv-4: There are two standard simplification rules

$$\underset{\sim}{g}(\tau_1,\ldots,\tau_p) \to \alpha_1 = \underset{\sim}{a} \ ,$$

and: $$\underset{\sim}{g}(\tau_1,\ldots,\tau_p) \to \alpha_2 = \underset{\sim}{b} \ .$$

Then, let $f \in pf_n(D^+)$ be an arbitrary partial function.

For any $\bar{\xi} \in (D^+)^n$, $\tilde{\alpha}(f,\bar{\xi}) = \tilde{\underset{\sim}{a}}(f,\bar{\xi}) = a$

$= \tilde{\underset{\sim}{b}}(f,\bar{\xi}) = b$, by

the observation at the end of § 5.2.1 .

Hence $\underset{\sim}{a} = \underset{\sim}{b}$ and $\gamma = \alpha_1 = \alpha_2$ will be suitable .

□

Comment: This is a Church-Rosser property for the standard simplifications. See Rosen (1971).

From this Lemma, we easily deduce an essential property of the standard simplifications:

Lemma: All standard simplifications of a term α terminate with the same term.

Proof: By induction on the size of the term α.

(i) $|\alpha| = 1$. Trivially true, since α has no standard simplification.

(ii) Assume true for all terms of size less than α, and assume that there are two standard simplifications of α:

$\alpha \overset{s}{\to} \alpha_1 \overset{s}{\to} \alpha_2 \overset{s}{\to} \ldots \overset{s}{\to} \alpha_n$, and:

$\alpha \overset{s}{\to} \alpha_1' \to \alpha_2' \overset{s}{\to} \ldots \overset{s}{\to} \alpha_n'$, .

$|\alpha_1| < |\alpha|$, and by induction hypothesis, all standard simplifications of α_1 terminate with the same term, which must be α_n. Similarly, all standard simplifications of α_1' terminate with the same term, which must be α_n', .

Now, by the previous Lemma, we know that there is a term γ such that $\alpha_1(\overset{s}{\rightarrow})^* \gamma$ and $\alpha_1'(\overset{s}{\rightarrow})^* \gamma$. But $|\gamma| < |\alpha|$, and therefore all the standard simplifications of γ must terminate with the same term γ_p. Now γ appears in a standard simplification of α_1, which implies $\alpha_n = \gamma_p$. Similarly $\alpha_{n'}' = \gamma_p$, from which we conclude: $\alpha_n = \alpha_{n'}'$.

□

For every term α free of $\bar{x}$, we will denote by $\mathrm{Simp}(\alpha)$ the common final term of all standard simplifications of α.

We will need one more property of standard simplifications, namely:

Lemma: Let α be a monotonically structured term, free of $\bar{x}$, such that $\tilde{\alpha}(\Omega, \ldots, \Omega) = \underset{\sim}{a}$ for some $a \in C$ (not ω). Then $\mathrm{Simp}(\alpha) = \underset{\sim}{a}$.

Proof: By structural induction on α.

Case i $\alpha = x_i$: excluded.

Case ii $\alpha = \underset{\sim}{a} \in C$. Then $\mathrm{Simp}(\alpha) = \underset{\sim}{a}$.

Case iii $\alpha = \underset{\sim}{\omega}$: excluded.

Case iv $\alpha = g(\alpha_1, \alpha_2, \ldots, \alpha_p)$:

$$\tilde{\alpha}(\Omega, \ldots, \Omega) = g(\tilde{\alpha}_1(\Omega, \ldots, \Omega), \ldots, \tilde{\alpha}_p(\Omega, \ldots, \Omega)) = \underset{\sim}{a}.$$

Let us denote $\tilde{\alpha}_i(\Omega, \ldots, \Omega)$ by $\underset{\sim}{a}_i$ for every i, $1 \leq i \leq p$.

We have $g(\underset{\sim}{a}_1, \ldots, \underset{\sim}{a}_p) = \underset{\sim}{a}$.

Now it is easy to see that, because of the morotonicity of g, there is a standard simplification schema:

$$g(\Lambda_1, \ldots, \Lambda_p) \rightarrow \underset{\sim}{a},$$

where, for every i, $1 \leq i \leq p$, if $\underset{\sim}{a}_i \neq \omega$ then $\Lambda_i = \underset{\sim}{a}_i$.

---

*/The notation $\tilde{\alpha}(f, \ldots, f)$, when α is free of $\bar{x}$, has been defined in §2.4.2.

[Let $\langle b_1,b_2,\ldots,b_p\rangle$ be an arbitrary element of Dom(g) such that $b_i = a_i$ for every position i, $1 \leq i \leq p$, such that $a_i \neq \omega$. To satisfy the definition of a standard simplification schema, we just have to show that $g(b_1,b_2,\ldots,b_p) = a$. But this is the case, because g is monotonic and $\langle a_1,a_2,\ldots,a_p\rangle \leq \langle b_1,b_2,\ldots,b_p\rangle$].

Now, by structural induction, for every position i, $1 \leq i \leq p$, such that $a_i \neq \omega$, we have $Simp(\alpha_i) = \underset{\sim}{a}_i$. Let us denote by β_i, for every i, $1 \leq i \leq p$, the term defined as follows:

$$\begin{cases} \text{if } a_i \neq \omega, & \beta_i = \underset{\sim}{a}_i , \\ \text{otherwise,} & \beta_i = \alpha_i . \end{cases}$$

Then we have the following:

$$\alpha = \underset{\sim}{g}(\alpha_1,\ldots,\alpha_p) \overset{s}{(\rightarrow)^*} \underset{\sim}{g}(\beta_1,\ldots,\beta_p) \overset{s}{\rightarrow} \underset{\sim}{a} ,$$

and since $\underset{\sim}{a}$ is the final term of this standard simplification of α, we have $Simp(\alpha) = \underset{\sim}{a}$, which proves the property for this case.

Case (γ) $\alpha = F(\alpha_1,\alpha_2,\ldots\alpha_n)$:

this case is excluded, because $\tilde{\tilde{\gamma}}(\Omega, \ ) = \omega$.

□

5.3 Full Computations */

5.3.1 Full substitutions

Let α be a term over some alphabet, and let $F(\overline{x}) \Leftarrow \tau(F,\overline{x})$ be a recursive definition. Then the result of substituting F by its definition for all occurrences of F in α, which we call full substitution, is a

*/ This section owes much to discussions of the author with J. Vuillemin.

term, denoted $\text{Fsubst}_\tau(\alpha)$, and which can be defined inductively in the obvious way:

(i) - (iii) $\alpha = x_i$, $\underset{\sim}{c}$, or $\underset{\sim}{w}$: $\text{Fsubst}_\tau(\alpha) = \alpha$;

(iv) $\alpha = \underset{\sim}{g}(\alpha_1,\ldots,\alpha_p)$:

$\text{Fsubst}_\tau(\alpha) = \underset{\sim}{g}(\text{Fsubst}_\tau(\alpha_1),\ldots,\text{Fsubst}_\tau(\alpha_p))$;

(v) $\alpha = F(\alpha_1,\ldots,\alpha_n)$:

$\text{Fsubst}_\tau(\alpha) = \tau(F,\langle\text{Fsubst}_\tau(\alpha_1),\ldots,\text{Fsubst}_\tau(\alpha_n)\rangle)$ [*].

If α is free of x, it is clear that one may go from α to $\text{Fsubst}_\tau(\alpha)$ by applying intermediate steps b2 of the definition of Computations in 2.4.2. This means that $\text{Fsubst}_\tau(\alpha)$ is allowed to be the term following α in a computation.

Full substitutions have the following interesting property:

Lemma: Let α be a correct term over some alphabet and let $F(\bar{x}) <= \tau(F,\bar{x})$ be a recursive definition. Then:

$\widetilde{\text{Fsubst}_\tau(\alpha)} = \tilde{\alpha} \cdot \tilde{\tau}$.

Proof: By definition of the functionals, the above statement is equivalent to:

$\forall f \in pf_n(D^+)$, $\widetilde{\text{Fsubst}_\tau(\alpha)}[f] = (\tilde{\alpha}\cdot\tilde{\tau})[f]$

$= \tilde{\alpha}[\tilde{\tau}[f]]$ Definition of functional composition.

i.e. $\forall f \in pf_n(D^+)$, $\forall \bar{\xi} \in (D^+)^n$, $\widetilde{\text{Fsubst}_\tau(\alpha)}[f](\bar{\xi}) = \tilde{\alpha}[\tilde{\tau}[f]](\bar{\xi})$

i.e. $\forall f \in pf_n(D^+)$, $\forall \bar{\xi} \in (D^+)^n$, $\widetilde{\text{Fsubst}_\tau(\alpha)}(f,\bar{\xi}) = \tilde{\alpha}(\tilde{\tau}[f],\bar{\xi})$.

We prove the latter statement by structural induction on α.

Case (i) $\alpha = x_i$, $1 \leq i \leq n$.

Then: $\widetilde{\text{Fsubst}_\tau(\alpha)}(f,\bar{\xi}) = \tilde{\alpha}(f,\bar{\xi}) \equiv \xi_i$,

and : $\tilde{\alpha}(\tilde{\tau}[f],\bar{\xi}) \equiv \xi_i$.

---

[*] The notation $\tau(F,\langle\beta_1,\ldots,\beta_n\rangle)$ has been defined formally in § 2.4.1.

Case (ii) $\alpha = \underset{\sim}{c} \in C$.

Then: $\widetilde{\text{Fsubst}_\tau(\alpha)}\ (f,\bar{\xi}) \equiv \tilde{\alpha}(f,\bar{\xi}) \equiv c$.

Case (iii) $\alpha = \underset{\sim}{v}$ Analogous to above.

Case (iv) $\alpha = \underset{\sim}{g}(\alpha_1,\alpha_2,\ldots,\alpha_p)$.

$$\widetilde{\text{Fsubst}_\tau(\alpha)}\ (f,\bar{\xi}) \equiv g(\widetilde{\text{Fsubst}_\tau(\alpha_1)}\ (f,\bar{\xi}),\ldots,\widetilde{\text{Fsubst}_\tau(\alpha_p)}\ (f,\bar{\xi}))$$

$$\equiv g(\tilde{\alpha}_1(\lceil f\rceil,\bar{\xi}),\ldots,\tilde{\alpha}_p(\lceil f\rceil,\bar{\xi})$$

$$\equiv \widetilde{\underset{\sim}{g}(\alpha_1,\ldots,\alpha_p)}\ (\lceil f\rceil,\bar{\xi})$$

$$\equiv \tilde{\alpha}(\lceil f\rceil,\bar{\xi}) .$$

Case (v) $\alpha = F(\alpha_1,\alpha_2,\ldots,\alpha_n)$.

$$\widetilde{\text{Fsubst}_\tau(\alpha)}\ (f,\bar{\xi}) \equiv \widetilde{\tau(F,\langle\text{Fsubst}_\tau(\alpha_1),\ldots,\text{Fsubst}_\tau(\alpha_n)\rangle)}\ (f,\bar{\xi})$$

$$\equiv \tilde{F}(f,\langle\widetilde{\text{Fsubst}_\tau(\alpha_1)}\ (f,\bar{\xi}),\ldots,\widetilde{\text{Fsubst}_\tau(\alpha_n)}\ (f,\bar{\xi})\rangle)$$

by Lemma 3.2.2 of Chapter 3,

$$\equiv \tilde{F}(f,\langle\tilde{\alpha}_1(\lceil f\rceil,\bar{\xi}),\ldots,\tilde{\alpha}_n(\lceil f\rceil,\bar{\xi})\rangle) .$$

On the other hand:

$$\tilde{\alpha}(\lceil f\rceil,\bar{\xi}) \equiv \widetilde{F(\alpha_1,\ldots,\alpha_n)}\ (\lceil f\rceil,\bar{\xi})$$

$$\equiv \lceil f\rceil\ (\tilde{\alpha}_1(\lceil f\rceil,\bar{\xi}),\ldots,\tilde{\alpha}_n(\lceil f\rceil,\bar{\xi}))\ ,\ \text{since } \alpha \text{ is correct.}$$

$$\equiv \tilde{F}(f,\langle\tilde{\alpha}_1(\lceil f\rceil,\bar{\xi}),\ldots,\tilde{\alpha}_n(\lceil f\rceil,\bar{\xi})\rangle)$$

$$\equiv \widetilde{\text{Fsubst}_\tau(\alpha)}\ (f,\bar{\xi}) \qquad \text{(above derivation).}$$

□

5.3.2 Full computations

Definition. The full computation of a term $\alpha(F,\overline{x})$ for $\overline{x} = \overline{c} \in (D^+)^n$, using a monotonically structured recursive definition:

$$F(\overline{x}) \Leftarrow \tau(F,\overline{x}) \ ,$$

is a sequence of terms α_i, such that:

$\alpha_0 = \alpha(F,\underset{\sim}{\overline{c}})$, and for every $k \geq 0$:

$$\begin{cases} \alpha_{2k+1} = \mathrm{Simp}(\alpha_{2k}) \\ \alpha_{2k+2} = \mathrm{Fsubst}_\tau(\alpha_{2k+1}) \ . \end{cases}$$

It is clear that a full computation is also a 'computation' in the sense of § 2.4.2, because, as we already observed, one can fill in intermediate steps between α_{2k} and $\mathrm{Simp}(\alpha_{2k})$ which satisfy case (b1) of the definition in § 2.4.2, and similarly between α_{2k+1} and $\mathrm{Fsubst}_\tau(\alpha_{2k+1})$ to satisfy case (b2).

5.3.3 Theorem 5

We will now show that the full computation actually leads to the least fixpoint of $\overset{\approx}{\tau}$ for a monotonically structured recursive definition $F(\overline{x}) \Leftarrow \tau(F,\overline{x})$.

We will first prove two lemmas.

Lemma: Let $\{\alpha_i \mid i \geq 0\}$ be the full computation of $\alpha(F,\overline{x})$ for $\overline{x} = \overline{c}$, using a monotonically structured recursive definition $F(\overline{x}) \Leftarrow \tau(F,\overline{x})$. Then, for every $k \geq 0$, for every $f \in pf_n(D^+)$, we have:

$$\overset{\approx}{\alpha}_{2k} = \overset{\approx}{\alpha}_{2k+1} = \alpha_0 \cdot \overset{\approx}{\tau}^k \ .$$

Proof: By induction on k.

(i) $k = 0$.

$\tilde{\tilde{\alpha}}_0 = \tilde{\tilde{\alpha}}_0 \cdot \tilde{\tau}^0$, since $\tilde{\tau}^0$ is the identity functional.

Then $\alpha_1 = \text{Simp}(\alpha_0)$ and therefore $\tilde{\tilde{\alpha}}_1 = \tilde{\tilde{\alpha}}_0$, by remark 1 of § 5.2.2.

(ii) Assume true for k-1, prove it for k.

(a) $\alpha_{2k+1} = \text{Simp}(\alpha_{2k})$, and therefore, by Remark 1 of § 5.2.2, since $\alpha_{2k} \xrightarrow{(S)^*} \alpha_{2k+1}$, we have:

$$\tilde{\tilde{\alpha}}_{2k} = \tilde{\tilde{\alpha}}_{2k+1}.$$

(b) $\alpha_{2k} = \text{Fsubst}_\tau(\alpha_{2k-1})$, and therefore, by the Lemma of Section 5.3, we have:

$$\tilde{\tilde{\alpha}}_{2k} = \widetilde{\text{Fsubst}_\tau(\alpha_{2k-1})} = \tilde{\tilde{\alpha}}_{2k-1} \cdot \tilde{\tau}$$

$$= \tilde{\tilde{\alpha}}_0 \cdot \tilde{\tau}^{k-1} \cdot \tilde{\tau}, \text{ by induction hypothesis.}$$

$$= \tilde{\tilde{\alpha}}_0 \cdot \tilde{\tau}^k, \text{ by associativity of functional composition and definition of } \tilde{\tau}^k. \quad \square$$

We can now prove the following Lemma which is of interest in its own right:

Lemma: Let α be a monotonically structured, correct term, $F(\bar{x}) \Leftarrow \tau(F,\bar{x})$ a monotonically structured recursive definition, and $\bar{c}$ an element of $(D^+)^n$. Let $\hat{f}_\tau$ denote the least fixpoint of $\tilde{\tau}$, and let a denote an element of Δ. Then:

(1) The full computation of α for $\bar{x} = \bar{c}$ using the recursive definition terminates with $\underset{\sim}{a}$ if and only if: $\tilde{\tilde{\alpha}}(\hat{f}_\tau, \bar{c}) = a$.

(2) The full computation of α for $\bar{x} = \bar{c}$ using the recursive definition does not terminate if and only if: $\tilde{\tilde{\alpha}}(\hat{f}_\tau, \bar{c}) = \omega$.

Proof: Note that (2) directly follows from (1) and from the fact that α is correct: $\tilde{\tilde{\alpha}}(\hat{\hat{f}}_\tau, \bar{c})$ is in Δ^+ and if it is not in Δ it must must be ω.

To prove (1), we first note that if the computation terminates with $\underset{\sim}{a}$, Lemma 3.2.3 immediately tells us that $\tilde{\tilde{\alpha}}(\hat{\hat{f}}_\tau, \bar{c}) \geq a$, since $\hat{\hat{f}}_\tau$ is a fixpoint of $\tilde{\tilde{\tau}}$.

So it remains to be shown that if $\tilde{\tilde{\alpha}}(\hat{\hat{f}}_\tau, \bar{c}) = a$, the full computation of α for $x = \bar{c}$ terminates with $\underset{\sim}{a}$.

We know that $\hat{\hat{f}}_\tau = \text{lub}\{\tilde{\tilde{\tau}}^i[\Omega] \mid i \geq 0\}$, because of Theorem 3 (Chapter 4, § 4.4.2).

Let $\{\alpha_i \mid i \geq 0\}$ be the full computation of α for $\bar{x} = \bar{c}$. Since α is monotonically structured, $\alpha_o = \alpha(F, \underset{\sim}{\bar{c}})$ is also monotonically structured, and therefore $\tilde{\tilde{\alpha}}_o$ is continuous over $mf_n(D^+)$ by § 4.3.4. But $\hat{\hat{f}}_\tau \in mf_n(D^+)$ by Theorem 3, and therefore we have:

$$\tilde{\tilde{\alpha}}_o[\text{lub}\{\tilde{\tilde{\tau}}^i[\Omega] \mid i \geq 0\}] = \text{lub}\{\tilde{\tilde{\alpha}}_o \cdot \tilde{\tilde{\tau}}^i[\Omega] \mid i \geq 0\},$$

i.e. $$\tilde{\tilde{\alpha}}_o[\hat{\hat{f}}_\tau] = \text{lub}\{\tilde{\tilde{\alpha}}_o \cdot \tilde{\tilde{\tau}}^i[\Omega] \mid i \geq 0\}.$$

But: $\forall \bar{\bar{f}} \in (D^+)^n$, $\tilde{\tilde{\alpha}}_o[\hat{\hat{f}}_\tau](\bar{\bar{f}}) = \tilde{\tilde{\alpha}}_o(\hat{\hat{f}}_\tau, \bar{\bar{f}})$

$= \tilde{\tilde{\alpha}}_o(\hat{\hat{f}}_\tau, \ )$, since α_o is free of $\bar{x}$,

$= \tilde{\tilde{\alpha}}(\hat{\hat{f}}_\tau, \bar{c})$

$= a \neq \omega$ (Hypothesis).

Therefore, by definition of the lub, for every $\bar{\bar{f}} \in (D^+)^n$, there must be an $i \geq 0$ such that:

$$\tilde{\tilde{\alpha}}_o \cdot \tilde{\tilde{\tau}}^i[\Omega](\bar{\bar{f}}) = a.$$

But, by the previous Lemma, we know that, for every $i \geq 0$,

$$\tilde{\tilde{\alpha}}_{2i} = \tilde{\tilde{\alpha}}_{2i+1} = \tilde{\tilde{\alpha}}_{o} \cdot \tilde{\tilde{\tau}}^{i}.$$

Hence, for every $\bar{\underset{\sim}{a}} \in (D^{+})^{n}$, there exists an $i \geq 0$ such that:

$$\tilde{\tilde{\alpha}}_{2i}[\Omega](\bar{\underset{\sim}{a}}) = \tilde{\tilde{\alpha}}_{2i}(\Omega, \bar{\underset{\sim}{a}}) = a.$$

But, by the Lemma of § 5.2.3, we know that this implies that $\mathrm{Simp}(\alpha_{2i}) = \underset{\sim}{a}$.

Hence $\alpha_{2i+1} = \mathrm{Simp}(\alpha_{2i}) = \underset{\sim}{a}$, which proves that the full computation of α for $\bar{x} = \bar{c}$ terminates with $\underset{\sim}{a}$. □

As a direct consequence of the previous Lemma, we get:

Theorem 5: For a monotonically structured recursive definition, the partial function over $(D^{+})^{n}$ computed by full computation is the least strong fixpoint of the recursive definition.

Proof: Let $F(\bar{x}) \Leftarrow \tau(F,\bar{x})$ be a monotonically structured recursive definition, let f_c be the partial function over $(D^{+})^{n}$ computed by full computation, and let $\hat{f}_{\tau}$ denote the least strong fixpoint of the recursive definition. By definition of f_c, if the full computation of $F(\bar{x})$ for $\bar{x} = \bar{c}$ terminates with $\underset{\sim}{a}$, then $f_c(\bar{c}) = a$. But, by the previous Lemma used for $\alpha = F(\bar{x})$, we also have $\hat{\hat{f}}_{\tau}(\bar{c}) = a$ in that case. If the full computation of $F(\bar{x})$ for $\bar{x} = \bar{c}$ does not terminate, $f_c(\bar{c}) = \omega$ by definition of f_c and $\hat{f}_{\tau}(\bar{c}) = \omega$ by the previous Lemma.

Hence: $f_c = \hat{f}_{\tau}$. □

5.4 Standard Innermost Computations

5.4.1 Parallel Innermost Substitutions

Let α be a term over some alphabet, and let $F(\bar{x}) \Leftarrow \tau(F,\bar{x})$ be a recursive definition. Then the result of substituting $F(\bar{\underset{\sim}{b}})$ by $\tau(F,\bar{\underset{\sim}{b}})$ where $\bar{\underset{\sim}{b}} \in C^{n}$, for all occurrences of such terms in α, which we call parallel innermost substitution, is a term denoted $\mathrm{Psubst}_{\tau}(\alpha)$,

which can be defined inductively in the obvious way:

(i) - (iii) $\alpha = x_i$, $\underset{\sim}{c}$ or $\underset{\sim}{y}$: $\quad Psubst_\tau(\alpha) = \alpha$.

(iv) $\alpha = \underset{\sim}{g}(\alpha_1, \ldots, \alpha_p)$:

$$Psubst_\tau(\alpha) = \underset{\sim}{g}(Psubst_\tau(\alpha_1), \ldots, Psubst_\tau(\alpha_p)).$$

(v) $\alpha = F(\alpha_1, \ldots, \alpha_n)$:

if, for every i, $1 \leq i \leq n$, $\alpha_i = \underset{\sim}{a}_i$, $\underset{\sim}{a}_i \in C$,

then: $Psubst_\tau(\alpha) = \tau(F, <\underset{\sim}{a}_1, \ldots, \underset{\sim}{a}_n>)$,

otherwise, $Psubst_\tau(\alpha) = F(Psubst_\tau(\alpha_1) \ldots\ldots, Psubst_\tau(\alpha_n))$.

If α is free of $\bar{x}$, it is clear that $Psubst(\alpha)$ can be derived from α by applying intermediate steps (b2) of the definition of 'elementary computations' in 2.4.2. Actually, these steps verify the conditions of 2.4.5, so $Psubst_\tau(\alpha)$ is allowed to be the term following α in an innermost computation.

5.4.2 Standard Innermost Computations

Definition: The standard innermost computation of a term $\alpha(F, \bar{x})$ for $\bar{x} = \bar{c} \in D^n$, using a recursive definition $F(\bar{x}) <= \tau(F, \bar{x})$, is a sequence of terms α_i such that:

$$\alpha_0 = \alpha(F, \bar{c}),$$

and, for every $k \geq 0$:

$$\begin{cases} \alpha_{2k+1} = Simp(\alpha_{2k}) \\ \alpha_{2k+2} = Psubst_\tau(\alpha_{2k+1}) \end{cases}$$, where the notation $Simp(\alpha)$ has been defined in § 5.2.3 and $Psubst_\tau(\alpha)$ in § 5.3.1.

It is clear that a standard innermost computation is an innermost computation in the sense of § 2.4.5.

5.4.3 **Theorem 6.**

We will now show that the Standard computation rule actually leads to the least fixpoint of $\tilde{\tau}$ for a monotonically structured recursive definition $F(\bar{x}) \Leftarrow \tau(F,\bar{x})$.

Let us denote by f_s the partial function over D^n which is the computed function of the recursive definition $F(\bar{x}) \Leftarrow \tau(F,\bar{x})$ by the standard computation rule.

We first prove:

Lemma: Let $\alpha(F,\bar{x})$ be a correct, monotonically structured term, and $F(\bar{x}) \Leftarrow \tau(F,\bar{x})$ be a monotonically structured recursive definition. Then, for every $\bar{c} \in D^n$:

1. if the standard innermost computation of α for $\bar{x} = \bar{c}$ using the recursive definition $F(x) \Leftarrow \tau(F,\bar{x})$ terminates with a constant term $\underset{\sim}{a} \in C$, then:

$$\tilde{\alpha}(f_s,\bar{c}) \equiv a,$$

2. otherwise: $\tilde{\alpha}(f_s,\bar{c}) \equiv \underset{\sim}{w}$.

Proof: We proceed by structural induction on α.

Cases (i)-(iii): $\alpha = x_i$, $\alpha = \underset{\sim}{a} \in C$, $\alpha = \underset{\sim}{w}$: trivial.

Case iv: $\alpha = \underset{\sim}{g}(\alpha_1,\ldots,\alpha_p)$.

Assume the Lemma true for every α_j, $1 \leq j \leq p$.

We have:

$\tilde{\alpha}(f_s,\bar{c}) \equiv g(\tilde{\alpha}_1(f_s,\bar{c}),\ldots,\tilde{\alpha}_p(f_s,\bar{c}))$ by 2.5.2 a(iv), since α is correct.

Let us designate $\tilde{\alpha}_j(f_s,\bar{c})$ by a_j, for every j, $1 \leq j \leq p$ and $g(a_1,\ldots,a_n)$ by b.

Because of the definition of the standard innermost computation,

we have $\alpha_i = \underset{\sim}{g}(\alpha_1^i,\ldots,\alpha_p^i)$ for some N, $0 \leq N \leq \infty$ and for every i, $0 \leq i \leq N$, where, for every j, $1 \leq j \leq p$, $\{\alpha_j^i \mid 0 \leq i \leq N\}$ consists of the N + 1 first elements of the standard innermost computation of α_j for $\overline{x} = \overline{c}$ using $F(\overline{x}) \Leftarrow \tau(F,\overline{x})$.

Only two cases may occur: N is infinite or N is finite.

Case iv-a: $N = \infty$

In this case, we must have $b \equiv \omega$. For assume $b \neq \omega$. Then let J be the set of indices between 1 and p such that the standard innermost computation of α_j for $\overline{x} = \overline{c}$ using $F(\overline{x}) \Leftarrow \tau(F,\overline{x})$ terminates. By induction hypothesis, we know that, for every such $j \in J$, the standard innermost computation of α_j terminates with $\underset{\sim}{a}_j$.

Now, let k be the smallest integer such that, for every $j \in J$, $\alpha_j^k = \underset{\sim}{a}_j$. (If J is empty, then k = 0; otherwise, k is the first integer for which all the terminating standard innermost computations of α_j's have actually terminated in k steps).

We have:

$\alpha_k = \underset{\sim}{g}(\alpha_1^k,\ldots,\alpha_p^k)$, where, for every $j \in J$, $\alpha_j^k = \underset{\sim}{a}_j$.

Summarizing what we know by induction hypothesis, we have, for for every j, $1 \leq j \leq p$:

if $j \in J$, then $\tilde{\alpha}_j(f_s,\overline{c}) = a_j \in D$;

if $j \notin J$, then $\tilde{\alpha}_j(f_s,c) = a_j \equiv \omega$.

But, since g is monotonic and $g(a_1,a_2,\ldots,a_p) = b \neq \omega$, there is a standard simplification schema:

$$\underset{\sim}{g}(A_1,A_2,\ldots A_p) \rightarrow \underset{\sim}{b},$$

where, for every $j \in J$, $A_j = \underset{\sim}{a}_j$.

Hence we have the simplification rule:

$$\underset{\sim}{g}(\alpha_1^k, \alpha_2^k, \ldots, \alpha_p^k) \rightarrow \underset{\sim}{b} ,$$

i.e.: $\alpha_k \rightarrow \underset{\sim}{b}$.

But this means that $Simp(\alpha_k) = \underset{\sim}{b}$.

Now if k is odd, this is a contradiction, because we must have $\alpha_k = Simp(\alpha_k)$,

If k is even, this means that $\alpha_{k+1} = \underset{\sim}{b}$, since $\alpha_{k+1} = Simp(\alpha_k)$ in this case. But this contradicts the assumption that the computation of α does not terminate.

Hence, the assumption $b \neq \omega$ was false.

In this case, we must have $\tilde{\alpha}(f_s, \bar{c}) \equiv \omega$ and the standard innermost computation of α does not terminate.

Case iv-b: $N < \infty$

We have: $\alpha_N = \underset{\sim}{g}(\alpha_1^N, \ldots, \alpha_p^N)$ and $\alpha_{N+1} = \underset{\sim}{a}$, since the computation of α terminates with $\underset{\sim}{a}$.

So there must be a standard simplification schema:

$$\underset{\sim}{g}(A_1, A_2, \ldots, A_p) \rightarrow \underset{\sim}{a} ,$$

of which $\underset{\sim}{g}(\alpha_1^N, \ldots, \alpha_p^N) \rightarrow \underset{\sim}{a}$ is an instance.

This means that:

(a) there must be an equation:

$$g(\xi_1, \xi_2 \ldots, \xi_p) \equiv a \qquad (E)$$

and a subset J of the indices from 1 to p such that E holds for every tuple $\langle \xi_1, \xi_2, \ldots, \xi_p \rangle \in Dom(g)$

such that $j \in J \Rightarrow f_j \equiv b_j$, where the b_j's are individual constants in Δ (not ω).

(b) For every $j \in J$, $\alpha_j^N = \underset{\sim}{b}_j$, where b_j is the constant defined in (a).

But (b) implies that, for every $j \in J$, the standard innermost computation of α_j for $\bar{x} = \bar{c}$ terminates with $\underset{\sim}{b}_j$. Hence, by the induction hypothesis, it must be that:

$$j \in J \Rightarrow \tilde{\alpha}_j(f_s, \bar{c}) \equiv b_j .$$

Now we know that:

$$\tilde{\alpha}(f_s, \bar{c}) \equiv g(\tilde{\alpha}_1(f_s, \bar{c}), \ldots, \tilde{\alpha}_p(f_s, \bar{c}))$$

$$\equiv g(a_1, \ldots, a_p), \text{ by definition of the } a_i\text{'s.}$$

Hence, the tuple $\langle a_1, \ldots a_p \rangle$ belongs to Dom(g), because α is assumed correct, and is such that:

$$j \in J \Rightarrow a_j \equiv b_j .$$

Thus equation (E) holds for this tuple and we have:

$$\tilde{\alpha}(f_s, \bar{c}) \equiv g(a_1, \ldots, a_p) \equiv a .$$

To summarize: in case iv-b: $\tilde{\alpha}(f_s, \bar{c}) \equiv a$ and the standard innermost computation of α terminates with $\underset{\sim}{a}$.

Cases iv-a and iv-b cover all the possibilities for case iv, so we have proved the Lemma for $\alpha = \underset{\sim}{g}(\alpha_1, \ldots \alpha_p)$.

Case v: $\alpha = F(\alpha_1, \alpha_2, \ldots, \alpha_n)$.

In this case:

if $\exists j$, $1 \leq j \leq n$, such that $\tilde{\alpha}_j(f_s, \bar{c}) \equiv \omega$, then $\tilde{\alpha}(f_s, \bar{c}) \equiv \omega$,

otherwise: $\tilde{\alpha}(f_s, \bar{c}) \equiv f_s(\tilde{\alpha}_1(f_s, \bar{c}), \ldots, \tilde{\alpha}_n(f_s, \bar{c}))$.

(By 2.3.2. a(v), since α is correct).

Because of the definition of the standard innermost computation of α, we must have:

$\alpha_i = F(\alpha_1^i, \ldots, \alpha_n^i)$ for some N, $0 \leq N \leq \infty$ and every i, $0 \leq i \leq N$, where, for every j, $1 \leq j \leq n$, $\{\alpha_j^i \mid 0 \leq i \leq N\}$ is the standard innermost computation of α_j for $\bar{x} = \bar{c}$ using $F(\bar{x}) \Leftarrow \tau(F, \bar{x})$.

Case v-a: If the standard innermost computation of α for $\bar{x} = \bar{c}$ terminates with $\underset{\sim}{a}$, then, for every j, $1 \leq j \leq n$, the standard innermost computation of α_j for $\bar{x} = \bar{c}$ must terminate with some $\underset{\sim}{a}_j$, $a_j \in D$ and the standard innermost computation of $F(\underset{\sim}{a}_1, \ldots, \underset{\sim}{a}_n)$ must terminate with $\underset{\sim}{a}$.

Hence, by induction hypothesis, $\forall j$, $1 \leq j \leq n$, $\tilde{\alpha}_j(f_s, \bar{c}) \equiv a_j$, and:

$$\tilde{\alpha}(f_s, \bar{c}) \equiv f_s(\tilde{\alpha}_1(f_s, \bar{c}), \ldots, \tilde{\alpha}_n(f_s, \bar{c})) .$$

$\equiv f_s(a_1,\ldots,a_n)$ (Induction hypothesis)

$\equiv a$ (Definition of $f_s(a_1,\ldots a_n)$).

Case v-b: If the standard innermost computation of α for $\bar{x} = \bar{c}$ does not terminate, then two subcases may appear:

v-b1. One of the standard innermost computations of α_j for $\bar{x} = \bar{c}$ does not terminate, for some j, $1 \leq j \leq n$. Then, by induction hypothesis, for that j, we have:

$$\tilde{\alpha}_j(f_s,\bar{c}) \equiv \omega .$$

Hence, $\tilde{\alpha}(f_s,\bar{c}) \equiv \omega$ by 2.3.2 a (v).

v-b2. For every j, $1 \leq j \leq n$, the standard innermost computation of α_j for $\bar{x} = \bar{c}$ terminates with $\underset{\sim}{a}_j \in C$, and the standard innermost computation of $F(\underset{\sim}{a}_1,\ldots,\underset{\sim}{a}_n)$ does not terminate. Because of the induction hypothesis, we have:

$\forall j$, $1 \leq j \leq n$, $\tilde{\alpha}_j(f_s,\bar{c}) = a_j \in D$. But since $\alpha = F(\alpha_1,\ldots,\alpha_n)$ is correct, we must have $\langle \tilde{\alpha}_1(f_s,\bar{c}),\ldots,\tilde{\alpha}_n(f_s,\bar{c}) \rangle \in (D^+)^n$.

Hence, $\forall j, 1 \leq j \leq n$, $\tilde{\alpha}_j(f_s,\bar{c}) \equiv a_j \in D$, and:

$\tilde{\alpha}(f_s,\bar{c}) \equiv f_s(\tilde{\alpha}_1(f_s,\bar{c}),\ldots,\tilde{\alpha}_n(f_s,\bar{c}))$

$\equiv f_s(a_1,\ldots,a_n)$ Induction hypothesis

$\equiv \omega$ Definition of $f_s(a_1,\ldots,a_n)$.

So in both subcases of case (v) where the standard innermost computation of α for $\bar{x} = \bar{c}$ does not terminate, we have $\tilde{\alpha}(f_s, \bar{c}) = \omega$.

This completes the proof for case (v) and for the Lemma. □

From this, we can now deduce:

Theorem 6: For a monotonically structured recursive definition, the partial function over D^n computed by the standard innermost computation is the least weak fixpoint of the recursive definition.

Proof: Let $F(\bar{x}) \Leftarrow \tau(F, \bar{x})$ by a monotonically structured recursive definition, and let f_s be the partial function computed by the standard innermost computation rule using this recursive definition. Let $\bar{c} \in D^n$.

The standard computation of $F(\bar{x})$ for $\bar{x} = \bar{c}$ starts as follows:

$\alpha_0 = F(\bar{c})$

$\alpha_1 = \text{Simp}(\alpha_0) = F(\bar{c})$

$\alpha_2 = \text{Psubst}_\tau(\alpha_1) = \tau(F, \bar{c})$

But the first term of the standard innermost computation of $\tau(F, \bar{x})$ for $\bar{x} = \bar{c}$ is $\tau(F, \bar{c})$.

Hence those computations either both terminate with the same term, say $\underset{\sim}{a}$, or both do not terminate.

The previous lemma applies, because $F(\bar{x})$ and $\tau(F, \bar{x})$ are both correct and monotonically structured, and thus we have:

If both computations terminate with $\underset{\sim}{a}$:

$$\widetilde{F(\bar{x})}(f_s,\bar{c}) \equiv f_s(\bar{c}) \equiv a,$$

$$\widetilde{\tau(F,\bar{x})}(f_s,\bar{c}) \equiv \tilde{\tau}(f_s,\bar{c}) \equiv a;$$

If both computations do not terminate:

$$f_s(\bar{c}) \equiv \omega ,$$

$$\tilde{\tau}(f_s,\bar{c}) \equiv \omega .$$

In any event, $f_s(\bar{c}) \equiv \tilde{\tau}(f_s,\bar{c})$.

Since $\bar{c}$ was arbitrary in D^n, this shows that f_s is a fixpoint of $\tilde{\tau}$.

Now, by Theorem 2, since f_s is a computed function obtained by innermost computation, we know that f_s must be the <u>least fixpoint of $\tilde{\tau}$</u>. □

<u>Comments</u>: 1. Notice that we have not used Theorem 4 in the proof of Theorem 6. In fact the proof of Theorem 6 is an alternate way of proving the existence of a least fixpoint of $\tilde{\tau}$. So far, the author has not been able to relate the proof of Theorem 6 with the fact that $\hat{f}_\tau = \text{lub}\ \{\tilde{\tau}^i[\Omega] \mid i \geq 0\}$, which is known by Theorem 4.

2. Theorem 6 tells us that $\hat{f}_\tau$ is a computed function. Therefore, by Theorem 1, we know that $\hat{f}_\tau$ must be an extension of $\hat{f}_\tau$, or rather:

$$\hat{f}_\tau^+ \prec \hat{f}_\tau ,$$

(Since, $\hat{f}_\tau \in pf_n(D)$ and $\hat{f}_\tau \in pf_n(D^+)$). This fact was not apparent from the set-theoretic characterizations of $\hat{f}_\tau$ and $\hat{f}_\tau$ provided in Theorems 3 and 4.

5.5 Safe Innermost Computations

This section describes a large class of innermost computation rules which also lead to the least weak fixpoint of the recursive definition, and which we call safe innermost. Both the standard simplification rules and the standard substitution rules are extended to provide more convenient and/or efficient rules to compute the least weak fixpoint . The standard innermost computation described in Section 5.5 appears as a special case of safe innermost computations. The safe innermost computation embody all previous methods known to us to lead to the least weak fixpoint.

We first describe the safe innermost computation rules, then prove Theorem 7 which says that they lead to the least weak fixpoint.

5.5.1 Safe Simplifications

A safe simplification schema is any expression of the form:

$$g(A_1, A_2, \ldots, A_p) \rightarrow B \ ,$$

where:

(i) each A_i is either an individual constant, or $\underset{\sim}{\omega}$, or a term variable;

(ii) B is either an individual constant, or $\underset{\sim}{\omega}$ or a term variable which is identical to A_j for some j, $1 \leq j \leq p$;

(iii) the equation $g(\xi_1, \xi_2, \ldots, \xi_p) \equiv \eta$ holds for all values of $< \xi_1, \xi_2, \ldots, \xi_p, \eta >$ such that:

(iii-1) $< \xi_1, \xi_2, \ldots, \xi_p > \ \varepsilon \ \ Dom(g)$;

(iii-2) For every i, $1 \leq i \leq p$, if A_i is a constant $\underset{\sim}{a}_i$ then $\xi_i = a_i$, if A_i is $\underset{\sim}{\omega}$ then $\xi_i = \omega$;

(iii-3) For every i,j, $1 \leq i, j \leq p$, if A_i and A_j are the same term variable, then $\xi_i \equiv \xi_j$;

(iii-4) If B is an individual constant $\underset{\sim}{b}$, then $\eta \equiv b$; if B is $\underset{\sim}{\omega}$ then $\eta \equiv \omega$, if B is the term variable A_j, $1 \leq j \leq p$, then $\eta = \xi_j$.

Comment: It is obvious from the definition that a standard simplification schema is a safe simplification schema. Notice that we don't exclude two term variables to be identical. Intuitively this means that one may decide to simplify on the basis that two terms are formally identical, if the corresponding equation holds.

A safe simplification rule is any correct instance of a safe simplification schema where all the term variables Ai's have been replaced by arbitrary terms τ_i's (in such a way that $\forall i, j, 1 \leq i, j \leq p$, if $A_i = A_j$ then $\tau_i = \tau_j$) and where B, if it is the term variable A_j, has been replaced by τ_j.

Examples:

(a) All standard simplification rules are also safe simplification rules.

(b) $\underset{\sim}{\text{if}}\ \underset{\sim}{T}\ \underset{\sim}{\text{then}}\ A\ \underset{\sim}{\text{else}}\ B \rightarrow A$, for the sequential 'if-then-else' connective, is a safe simplification schema which is not a standard one (cf example (c) in § 5.2.1)

(c) $\underset{\sim}{\text{if}}\ A\ \underset{\sim}{\text{then}}\ B\ \underset{\sim}{\text{else}}\ B \rightarrow B$, for the parallel 'if-then-else' connective, is a safe simplification schema which is not a standard one (cf example (d) in § 5.2.1). It corresponds to the equation:

$$\forall \xi_1 \in \{T,F\}^+, \forall \xi_2 \in D^+ : \text{if } \xi_1 \text{ then } \xi_2 \text{ else } \xi_2 = \xi_2,$$

valid for the parallel 'if-then-else' connective.

(d) If '*' is the ordinary binary multiplication over the integers extended by $0 * \omega = \omega$, we have seen in Example (g) of 5.2.1 that $0 * A \rightarrow 0$ was not a standard simplification schema, because the corresponding equation did not hold on the whole Domain of '*'. For the same reason, it cannot be a safe simplification schema either.

A <u>safe set of simplification rules</u> is any set of simplification rules which contains the set of standard simplification rules.

5.5.2 <u>Safe Innermost Substitutions</u>

The safe innermost substitutions are innermost substitutions performed on certain key positions, which we are now going to define:

<u>Definition</u>: Let $g(\alpha_1, \alpha_2, \ldots, \alpha_p)$ be a correct term. Let J be the set of indices corresponding to constant terms, i.e.

$$J = \{ j \mid 1 \leq j \leq p \text{ and } \alpha_j = a_j \in C \}.$$

An <u>ω-set</u> I of the given function g in that term is a set of indices such that the equation:

$$g(\xi_1, \xi_2, \ldots \xi_p) = \omega$$

holds for every tuple $< \xi_1, \xi_2, \ldots, \xi_p > \in \text{Dom}(g)$ such that

(1) $\forall j \in J, \ \xi_j = a_j$,

(2) $I \cap J = \emptyset$ and $\forall i \in I, \ \xi_i = \omega$.

In other words, an ω-set is a subset of the indices such that, if all the corresponding arguments are undefined, then the whole term becomes undefined, for all values of the other arguments respecting the constant terms.

Notice that the union of two ω-sets of g in a term is also an ω-set of g in the term.

We now proceed to define inductively sets of key occurrences of F in which to perform safe innermost substitutions in a correct term α:

(i) - (iii) $\alpha = x_i$, $\underset{\sim}{c}$ or $\underset{\sim}{\omega}$: no substitution to perform.

(iv) $\alpha = g(\alpha_1, \alpha_2, \ldots, \alpha_p)$: select an g-set I of g in α, as discussed above. To obtain a set of key occurrences of F in α, pick a set of key occurrences of F in each of the α_i's for every $i \in I$, and form the union of them.

(v) $\alpha = F(\alpha_1, \ldots, \alpha_n)$:

-if all α_i, for $1 \leq i \leq n$ are individual constants in C then this occurrence of F is a key occurrence.

-otherwise, to obtain a set of key occurrences of F in α, pick a set of key occurrences of F in each of the α_i's, for $1 \leq i \leq n$, and form the union of them.

Comment:

1) Notice that this is not a deterministic process. In general, there will be many sets of key occurrences of F in a given term, according to which g-set of the g's we pick during the process.

2) Notice that in Case v we do not select an occurrence of F corresponding to a term $F(\alpha_1, \ldots, \alpha_n)$ in which some of the α_i's are not individual constants (even though the α_i's might not contain F). Also, an α_i being $\underset{\sim}{\omega}$ prevents the F of $F(\alpha_1, \ldots, \alpha_n)$ from being a key occurrence. This is of minor importance, however, since the definition of an innermost computation would not permit substitution in such a position, even if we had included it in the set of key positions.

Example: Take α = if P(F,x) then A(F,x) else B(F,x).

(a) Suppose that 'if-then-else' is the sequential connective.

Then it has five ω-sets in α:

$\{1\}, \{1,2\}, \{1,3\}, \{2,3\}, \{1,2,3\}$.

So any set of key-occurrences of F in $P(F,x)$ will be a set of key-occurrences of F in α; similarly, the union of any set of key-occurrences of F in $A(F,x)$ with any set of key-occurrences of F in $B(F,x)$ will be a set of key-occurrences of F in α, etc...

(b) Suppose now that 'if-then-else' is the parallel connective. It only has four ω-sets now in α, namely:

$\{1,2\}, \{1,3\}, \{2,3\}, \{1,2,3\}$.

Suppose that we have picked $\{2,3\}$ and that after some computation, the term α has become:

$\beta = \underset{\sim}{\text{if}}\ P(F,x)\ \underset{\sim}{\text{then}}\ \underset{\sim}{a}\ \text{else}\ B(F,x)$

Now, 'if-then-else' only has one ω-set in β, namely $\{1,3\}$. (This means intuitively that computing in position 1 alone or in position 3 alone would be unsafe).

5.5.3 Safe Innermost Computations

Definition: A safe innermost computation of a correct term α for $\overline{x} = \overline{c} \in D^n$ using the recursive definition $F(\overline{x}) \Leftarrow \tau(F,\overline{x})$ is a sequence of terms $\{\alpha_i \mid i \geq 0\}$ such that:

(a) $\alpha_0 = \alpha(F,\underset{\sim}{c})$;

(b) For every $k \geq 0$:

(b1) α_{2k+1} is obtained from α_{2k} by selecting a safe set of simplifications and applying

them, in any order, until no more can be applied.

(b) α_{k+2} is obtained from α_{k+1} by performing innermost substitutions in a set of key occurrences of F in α_{k+1}.

Comment:

In general, there are many safe innermost computations of a given term, since at each step, we have the choice of which safe set of simplifications to pick, and which set of key occurrences of F to pick. These choices may vary from step to step, and may even be context sensitive, i.e., one may very well decide to make the choice of key occurrences of F in a term on the basis of the environment of that term.

The rest of the Chapter is devoted to prove that any computed function computed by safe innermost computation is the least weak fixpoint of the recursive definition.

5.5.4 Theorem 2

Subcomputations

Definition: Let $\{\alpha_i \mid i \geq 0\}$ be an innermost computation. A subcomputation of this computation is a sequence of consecutive subterms of α_i, for $N_1 \leq i \leq N$, which form a computation sequence.

N can be finite or infinite, and accordingly, the subcomputation will be finite or infinite. A subcomputation is said to terminate if it reaches a term which is an individual constant in C. A subcomputation can be finite and not terminate, in the case where the position corresponding to the subcomputation disappears.

For example, if:

$$g(\alpha_1,\beta_1) \to g(\alpha_2,\beta_2) \to g(\alpha_3,\beta_3) \to \beta_3 \to g(\alpha_4,\beta_4) \to \ldots.$$

is an innermost computation, then:

$\{\beta_1, \beta_2, \beta_3, \beta_4, g(\alpha_4,\beta_4), \ldots\}$ and $\{\alpha_1, \alpha_2, \alpha_3\}$ are subcomputations. The second one is finite, but may not have terminated if α_3 is not an individual constant in C.

A well-founded ordering on D^n

We now give a well-founded, strict partial ordering*/ on D^n, denoted $<_1$ by:

Definition:

$\forall \bar{\xi}, \bar{\eta} \in D^n$, $\bar{\xi} <_1 \bar{\eta}$ if the standard innermost computation of $F(\bar{\eta})$ terminates and the standard innermost computation of $F(\bar{\xi})$ is a subcomputation of it which terminates.

Comment:

Let us emphasize that the standard innermost computation of $F(\bar{\xi})$ must terminate as a subcomputation of $F(\bar{\eta})$. It would not be sufficient to require that $F(\bar{\xi})$ occurred within the standard innermost computation of $F(\bar{\eta})$, and that its standard innermost computation terminate. If the latter did not terminate inside that of $F(\bar{\eta})$, we could not guarantee antisymmetry of the

---

*/ A binary relation $<$ over a set S is a strict partial ordering if it is antisymmetric:

$$(\forall a,b)\ (a < b \text{ and } b < a \text{ is false}),$$

and transitive:

$$(\forall a,b,c)\ (a < b \text{ and } b < c \Rightarrow a < c).$$

A strict partial ordering is well-founded if it has no infinite decreasing chain.

relation.

Lemma: The relation $<$ defined above is a well founded, strict partial ordering on D^n.

Proof:

(i) Antisymmetry: Trivial since $\bar{\xi} < \bar{\tau}$ implies that the standard innermost computation of $F(\bar{\tau})$ is strictly shorter than that of $F(\bar{\tau})$.

(ii) Transitivity: Trivial.

(iii) Well-foundedness: Suppose we have an infinite descending chain: $\tau_0 > \tau_1 > \tau_2 > \ldots$. Since each term is finite, there must be an infinite number of distinct terms in the computation of $F(\bar{\tau}_0)$ to contain the infinity of $F(\bar{\tau}_i)$. Hence the computation of $F(\bar{\tau}_0)$ does not terminate, which contradicts the hypothesis $\tau_1 < \tau_0$. =

We also have a well-founded strict partial ordering on the set of terms over an alphabet, namely the subterm relation. Let us also denote it by $<$.

Therefore, we can form a well founded strict partial ordering on $D^n \times \{\text{Terms}\}$, which we will also denote $<$, as the lexicographical ordering obtained from the two previous ones, namely:

$\forall < \bar{\xi}, \alpha >, < \bar{\tau}, \beta > \in D^n \times \{\text{Terms}\}$:

$< \bar{\xi}, \alpha > < < \bar{\tau}, \beta >$ iff $\bar{\xi} < \bar{\tau}$ or $(\bar{\xi} = \bar{\tau}$ and $\alpha < \beta)$.

It is a well known result that such a lexicographical ordering is indeed a well-founded strict partial ordering if it is obtained from orderings which have the same property.

We will need one more definition:

A subset S of $(\Delta^+)^p$ is said to be regular iff for every $\bar{\xi} \in S$,

every $\bar{\eta}$ in $(\Delta^+)^p$ which is $\prec \bar{\xi}$ is also in S. An interpretation of an alphabet is regular iff all the given functions have regular domains. Similarly, a recursive definition is regular iff all the given functions which appear in it have regular domains.

Any interpretation can be extended into a regular one in a number of ways, and any monotonic interpretation can also be extended into a monotonic regular one in a number of ways. Now it is easy to see that if we take a monotonically structured recursive definition which is not regular and extend the given functions into monotonic regular ones, the least weak fixpoint of the recursive definition is not modified. (Because if the recursive definition is $F(\bar{x}) \Leftarrow \tau(F,\bar{x})$, $\tau(F,\bar{x})$ must be correct; therefore, evaluating $\tau(F,\bar{x})$ for any partial function $f \in pf_n(D)$ cannot lead outside the domain of the given functions).

We now have the necessary tools to prove:

Lemma: Let the interpretation of the alphabet be regular, and let $F(\bar{x}) \Leftarrow \tau(F,\bar{x})$ be a monotonically structured recursive definition. Then for every $\bar{\xi} \in D^n$ and for every correct monotonically structured term α free of $\bar{x}$, if the standard innermost computation of $F(\bar{\xi})$ terminates and has the standard innermost computation of α as a terminating subcomputation, then any safe innermost computation of α terminates.

Proof: We use structural induction on the pair $\langle \bar{\xi}, \alpha \rangle$, i.e., we prove that, if the property holds for all pairs less than $\langle \bar{\xi}, \alpha \rangle$, then it holds for $\langle \bar{\xi}, \alpha \rangle$.

We proceed by case analysis on α.

Cases (i) - (iii): $\alpha = x_i$, $\underset{\sim}{c}$ or $\underset{\sim}{\omega}$. The property is trivial, because in Case (i) - (ii) any computation of α terminates at the first step and in Case (iii) no computation of α terminates.

Case (iv): $\alpha = \underset{\sim}{g}(\alpha_1,\ldots,\alpha_p)$.

Let us pick a safe innermost computation of α and assume it does not terminate. Two cases can arise.

Case (iv-1): $\alpha_i = \underset{\sim}{g}(\alpha_1^i,\ldots,\alpha_p^i)$ for every $i \geq 0$.

Let us take some N sufficiently large so that all the terminating subcomputations of α_j's have terminated before N, say $N \geq i_0$. Let I be the set of indices such that the subcomputations of α_j, $j \in I$ terminate, with, say, $\underset{\sim}{a}_j$. Then, for every j in I, $\alpha_j^N = \underset{\sim}{a}_j$.

We have $\alpha_N = \underset{\sim}{g}(\alpha_1^N,\ldots,\alpha_p^N)$.

Now g has an ω-set in α_N. For example, we have:

$g(\xi_1,\ldots,\xi_p) = \omega$ where $\begin{cases} j \in I \Rightarrow \xi_j = a_j & */ \\ j \notin I \Rightarrow \xi_j = \omega \ . \end{cases}$

For if $g(\xi_1,\ldots,\xi_p) = a \neq \omega$, we would have a standard simplification rule $g(\alpha_1^N,\ldots,\alpha_p^N) \to \underset{\sim}{a}$.

---

*/ This $\langle \xi_1,\ldots,\xi_p \rangle$ belongs to the Domain of g: This comes from the regularity of $Dom(g)$ and the fact that α is correct. If $\hat{f}_\tau$ denotes the least fixpoint of $\tilde{\tau}$, we know that $\tilde{\alpha}(\hat{f}_\tau,) \equiv g(a_1,\ldots,a_p)$, where $a_i = \tilde{\alpha}_i(\hat{f}_\tau,)$; hence $\langle a_1,\ldots,a_p \rangle \in Dom(g)$ (α correct). But $\langle \xi_1,\ldots,\xi_p \rangle \leq \langle \alpha_1,\ldots,\alpha_p \rangle$: hence $\langle \xi_1,\ldots,\xi_p \rangle \in Dom(g)$ (Regularity).

Now the set of safe simplification rules must include that particular rule. So either of two things can happen:

(i) this particular rule is applied.

(ii) Another rule is applied, say $g(\alpha_1^N,\ldots,\alpha_p^N) \rightarrow \alpha_j^N$

In both cases, and without further analysis, the hypothesis of Case (iv-1) is violated.

So, in fact, for $N \geq i_0$, there is an ω-set of g in α_N.

Let us denote by J the union of those ω-sets of g in α_N which are selected infinitely many times in the safe innermost substitution process. J is also an ω-set of g in α_N, and therefore, we have an equation:

$$g(\xi_1,\ldots,\xi_p) \equiv \omega \qquad \text{(E1)}$$

for every tuple $\langle \xi_1,\ldots,\xi_p \rangle \in \mathrm{Dom}(g)$ such that:

$$\forall j \in I,\ \xi_j \equiv a_j$$

$$\forall j \in J,\ \xi_j \equiv \omega.$$

Now, since the standard innermost computation of α terminates, we must have an equation:

$$g(\xi_1,\ldots,\xi_p) \equiv a \quad \text{for some } a \neq \omega, \qquad \text{(E2)}$$

for every tuple $\langle \xi_1,\ldots,\xi_p \rangle \in \mathrm{Dom}(g)$ such that:

$$\forall j \in K,\ \xi_j \equiv a_j,$$

where K is the set of indices j for which the standard innermost computation of α_j terminates as a subcomputation of the standard innermost computation of α.

The key point is that $K \cap J = \emptyset$. For suppose there is an index $j \in K \cap J$. Then, since $j \in K$, the standard innermost computation of α_j terminates as a subcomputation of that of α, hence, by transitivity, also

as a subcomputation of $F(\bar{\xi})$. But $\langle \bar{\xi}, \alpha_j \rangle \prec \langle \bar{\xi}, \alpha \rangle$, and thus, by induction, we know that any safe innermost computation of α_j must terminate. But, since $j \in J$, the subcomputation of α_j in the safe innermost computation of α does not terminate. However, α_j is worked upon infinitely many times, hence this subcomputation is also a safe innermost computation. This is a contradiction.

Because K and J are disjoint, and because of the regularity of Dom(g), we can select a tuple $\langle c_1, c_2, \ldots, c_p \rangle$ which will satisfy both (E1) and (E2), by specifying:

$$\begin{cases} \forall j \in J, \ c_j \equiv \omega \\ \forall j \in I \cup K, \ c_j \equiv a_j \\ \text{otherwise} \ c_j \equiv \omega. \end{cases}$$

(If I and K are not disjoint, which means that, for some α_j two computations of α_j have terminated, we know that the final term must be the same in each case, $\underset{\sim}{a}_j$).

$\langle c_1, c_2, \ldots, c_p \rangle \prec \langle a_1, \ldots, a_p \rangle$ and therefore, $\langle c_1, \ldots, c_p \rangle$ belongs to Dom(g), by regularity (See footnote on previous page). Hence $\langle c_1, c_2, \ldots, c_p \rangle$ satisfies E1 and E2, which is a contradiction.

Therefore this case cannot happen.

<u>Case (iv-2)</u> $\gamma_i = \underset{\sim}{g}(\alpha_1^i, \ldots, \alpha_p^i)$ for $0 \leq i \leq N$ and α_{N+1} is the final term of a safe simplification of α_j^N for some j, or $\underset{\sim}{\omega}$. (It cannot be an individual constant, because the computation of α would then terminate).

So there must be a simplification scheme

$$\underset{\sim}{g}(A_1, A_2, \ldots, A_p) \rightarrow B$$

corresponding to an equation

$$g(\xi_1,\xi_2,\ldots,\xi_p) \equiv \eta\,, \qquad (E3)$$

where:

$$\begin{cases} <\xi_1,\ldots,\xi_p> \in \mathrm{Dom}(g) \\ \forall j,\ 1 \le j \le p,\quad \begin{cases} A_j = \underset{\sim}{a}_j \Rightarrow \xi_j \equiv a_j \\ A_j = \underset{\sim}{\omega} \Rightarrow \xi_j \equiv \omega. \end{cases} \\ \forall j,k,\ 1 \le j,k \le p,\quad A_j = A_k \Rightarrow \xi_j \equiv \xi_k \\ \text{if} \quad B = A_j \quad \text{then} \quad \eta \equiv \xi_j \\ \text{if} \quad B = \underset{\sim}{\omega} \quad \text{then} \quad \eta \equiv \omega. \end{cases}$$

Let us call L the set of positions j such that the subcomputations of α_j have terminated in the safe innermost computation of α. For $j \in L$, we have $\xi_j \equiv a_j$.

As in the previous case equation (E2) must also hold. K denotes, as in the previous case, the set of positions such that the standard subcomputations of α_j have terminated in the standard computation of α.

Now let us choose $<d_1,\ldots,d_p>$ in the following way:

$$\begin{cases} \forall j \in L \cup K,\quad d_j \equiv a_j \\ \forall j \notin (L \cup K) \text{ if } \alpha_j^N = \alpha_k^N \text{ for some } k \in K, \text{ then } d_j \equiv a_k \\ \text{otherwise,}\quad d_j \equiv \omega. \end{cases}$$

We observe that if $\alpha_j^N = \alpha_k^N$ and $k \in K$, then the standard innermost computation of α_j must terminate. (The argument is as follows: since $k \in K$, the standard innermost computation of α_k terminates, and, by induction hypothesis any safe innermost computation of α_k terminates. Hence, that which leads from α_k to α_k^N must also terminate, and so does the one which leads from α_j to $\alpha_j^N = \alpha_k^N$. But if a safe innermost computation of α_j terminates, a fortiori the standard innermost one will.)

Hence $d_k \equiv a_k$, and we observe that:

$< d_1,\ldots,d_p > \prec < a_1,\ldots,a_p >$ and therefore

$< d_1,\ldots,d_p > \in \mathrm{Dom}(g)$, and $< d_1,\ldots,d_p >$ satisfies (E2).

Now we show that $< d_1,\ldots,d_p >$ also satisfies (E3) with $\eta \equiv \omega$.

(a) First, if $j \in L$ $d_j \equiv a_j$.

(b) If $\alpha_j^N = \underset{\sim}{\omega}$, α_j^N cannot be equal to α_k^N for $k \in K$, by the above argument (the standard innermost computation of $\underset{\sim}{\omega}$ never terminates), and a fortiori j cannot be in K.

So $d_j \equiv \omega$.

(c) Suppose $\alpha_j^N = \alpha_k^N$.

(c1) If either j or k is in L, then the other is, and we have $d_j \equiv d_k \equiv a_j \equiv a_k$. So assume now that $j \notin L$ and $k \notin L$:

(c2) If $j \in K$ and $k \in K$, then both standard innermost computations of α_j and α_k terminate with the same result, and we have $d_j \equiv d_k \equiv a_j \equiv a_k$.

(c3) If $j \in K \cup L$ and $k \in K$ (or the symmetric case), both standard innermost computations of α_j and α_k terminate with the same result (by the same argument used in the proof that $<d_1,\ldots,d_p>$ satisfies (E2)), and we have forced $d_j \equiv a_k \equiv d_k$ in the definition of the d_j's.

(c4) If $j \notin K \cup L$ and $k \notin K \cup L$. Then either there is an ℓ in K such that $\alpha_j^N = \alpha_\ell^N$ (in which case also $\alpha_\ell^N = \alpha_k^N$ and $d_j \equiv d_\ell$, $d_p \equiv d_\ell$, so $d_j = d_k$) or there is no such ℓ, in which case $d_j \equiv d_k \equiv \omega$.

So $< d_1,d_2,\ldots,d_p >$ certainly satisfies (E3). Now to show that

the corresponding right hand side in E3 is ω, we assume B is not $\underset{\sim}{\omega}$ (in which case it is obvious); then $B = \alpha_v^N$ for some v, $1 \leq v \leq p$.

First, <u>v does not belong to L ∪ K</u>: v is not in L because otherwise B would be a constant, which is excluded. It is not in K either because, by induction hypothesis, the safe innermost computation of α_v would then terminate: but then so would that of α_v^N, hence that of α_{N+1} hence that of α, which is a contradiction.

Furthermore, there is no $k \in K$ such that $\alpha_k^N = \alpha_v^N$, for otherwise the standard innermost computation of α_v would terminate, by a previously seen argument, and again this would imply, as above, that the safe innermost computation of α terminates, which is a contradiction.

Therefore $d_v \equiv \omega$, which implies $\eta \equiv \omega$.

Thus, we see that:

$$g(d_1,\ldots,d_p) \equiv \omega \text{ from } (E3)$$
$$\equiv a \not\equiv \omega \text{ from } (E2),$$

which is a contradiction.

This proves the property for Case iv.

<u>Case (v)</u>: $\alpha = F(\alpha_1,\ldots,\alpha_n)$

If the standard innermost computation of α terminates with $\underset{\sim}{a}$, then for every i, $1 \leq i \leq n$, that of α_i terminates (with $\underset{\sim}{a}_i$), and that of $F(\underset{\sim}{a}_1,\ldots,\underset{\sim}{a}_n)$ does, with $\underset{\sim}{a}$.

Any safe innermost computation of α is of the form $\alpha_i = F(\alpha_1^i,\ldots,\alpha_n^i)$, in which all the computations of α_j, for $1 \leq j \leq n$, terminate with $\underset{\sim}{a}_i$, by the induction hypothesis.

Now $< a_1,\ldots,a_n >$ is less than $< \xi_1,\ldots,\xi_n >$ in our ordering on D^n,

since the standard innermost computation of $F(\underset{\sim}{a}_1,\ldots,\underset{\sim}{a}_n)$ occurs as a terminating subcomputation of the standard innermost computation of $F(\underset{\sim}{\xi}_1,\ldots,\underset{\sim}{\xi}_n)$.

Let us abbreviate, as usual, $<\underset{\sim}{a}_1,\ldots,\underset{\sim}{a}_n>$ by $\underset{\sim}{\bar{a}}$. Then the pair $<\underset{\sim}{\bar{a}}, F(\underset{\sim}{\bar{a}})>$ is strictly less than the pair $<\underset{\sim}{\bar{\xi}},\alpha>$ in our order, since $\underset{\sim}{\bar{a}} < \underset{\sim}{\bar{\xi}}$. Hence by induction hypothesis, our property is true for the pair $<\underset{\sim}{\bar{a}}, F(\underset{\sim}{\bar{a}})>$. Since the standard innermost computation of $F(\underset{\sim}{\bar{a}})$ terminates and obviously has $F(\underset{\sim}{\bar{a}})$ itself as a terminating subcomputation, we know that any safe innermost computation of $F(\underset{\sim}{\bar{a}})$ will terminate.

But any safe innermost computation of α hits upon a safe innermost computation of $F(\underset{\sim}{\bar{a}})$, hence also terminates.

This disposes of Case v and completes the proof. □

From the Lemma, we can conclude:

Theorem 7: For a monotonically structured recursive definition, the partial function over D^n computed using safe innermost computation for any regular monotonic extension of the recursive definition is the least weak fixpoint of the recursive definition.

Proof: As observed above, if the recursive definition is not regular, one can extend it into a monotonic regular one without modifying the least weak fixpoint.

Let f_{safe} be any computed function of the recursive definition using safe innermost computations, and f_s be the function computed with the standard innermost rule.

Applying the above Lemma with $\alpha = F(\underset{\sim}{\bar{\xi}})$ immediately gives:

$$f_s \leq f_{safe}.$$

Now, since $f_s = \hat{f}_\tau$ by Theorem 6, ($\hat{f}_\tau$ denotes the least fixpoint of $\tilde{\tau}$), and since f_{safe} is an innermost computed function, Theorem 2 shows that $f_{safe} = \hat{f}_\tau$. □

CONCLUSION

In this work, we have presented a model for 'recursive definitions' which enabled us to prove general results on the relation between computed functions and fixpoints, and we have given some fixpoint computation rules for a certain class of recursive definitions.

We now suggest some possible directions in which further research could possibly extend this work:

(a) In Chapter 5, in addition to the standard innermost computation rule we gave a wide class of computation rules, the safe innermost rules, which also lead to the least weak fixpoint of the (monotonically structured) recursive definition. We have given no such rules for the least strong fixpoint, in addition to the full computation rule. There is clearly a need for such rules, and as a matter of fact Vuillemin (1972) has recently found such rules, which he calls "safe computation rules"; he also gives conditions for such a rule to be optimal.

(b) The domain D^+ of our interpretation has a particularly simple structure: the partial order on D^+ has only two levels so to speak, with ω being less defined than every element in D, and every element in D only related to itself.

It would be interesting to see how the computational part of the theory extends to domains S with a much richer structure, for example partially ordered sets which have a least element and are chain-closed (in order to guarantee existence and good properties of fixpoint of continuous functionals).

(c) Scott (1969) presents a mathematical theory of computation of higher types of objects. He suggests that it would be worthwhile to establish that the computable objects in his sense are also computable in the usual sense of Church's Thesis. The present work establishes the fact for objects of low types (partial functions and functionals). It might be of interest to investigate if and how the computation methods presented here could be extended to objects of higher types.

(d) Syntactic extensions of the 'recursive definitions' might be interesting to investigate. For example, we might consider an expression of the form:

$$\alpha(F,\bar{x}) \simeq \beta(F,\bar{x}) \quad ,$$

where α and β are two arbitrary terms, to represent a recursive definition. In the case where such definitions possess fixpoints, is there a general computation mechanism that will compute one (or several) of them?

APPENDIX I

MONOTONICITY, CONTINUITY. APPLICATIONS TO PARTIAL FUNCTIONS

A Introduction

B Partially ordered sets

- B.1 General Definitions
- B.2 Monotonicity, Continuity
- B.3 Ordinals
- B.4 Least Upper Bounds
  - B.4.1 Monotonic Mappings
  - B.4.2 Continuous Mappings
- B.5 Greatest Lower Bounds

C Partial Functions and the Extension Relation

- C.1 Definitions
- C.2 Properties of Sets of Partial Functions
- C.3 Properties of Sets of Monotonic Partial Functions

A. Introduction

In this Appendix, we derive fixpoint properties of monotonic and continuous mappings over partially ordered sets. We then apply these properties to sets of partial functions, after verifying that these sets indeed satisfy the required conditions.

The first result on monotonic mappings is a slight generalization of a result mentioned in an unpublished paper by Park (1970). The result on continuous mappings generalizes a Theorem due to Kleene (1952). The second result on monotonic mappings is a mild generalization of a result due to Tarski (1955). Our proof follows that given by Park (1969). Bekić (1969) and Scott (1970) contain related work.

B. Partially ordered sets

B.1 General Definitions

An ordering on a set M (or partial ordering on M) is a binary relation on M, denoted $\preccurlyeq$, which is:

(i) reflexive, i.e.: $\forall x \in M,\ x \preccurlyeq x$,

(ii) antisymmetric, i.e.: $\forall x,y \in M,\ x \preccurlyeq y \wedge y \preccurlyeq x \Rightarrow x = y$,

(iii) transitive, i.e.: $\forall x,y,z \in M,\ x \preccurlyeq y \wedge y \preccurlyeq z \Rightarrow x \preccurlyeq z$.

A strict ordering on M (or strict partial ordering on M) is a binary relation on M, denoted $\prec$, which is:

(i) strictly antisymmetric, i.e.:

$\forall x,y \in M, x \prec y \wedge y \prec x$ is false,

(ii) transitive, i.e. $\forall x,y,z \in M,\ x \prec y \wedge y \prec z \Rightarrow x \prec z$.

It is easy to transform a strict ordering on a set M into an ordering on M by adding the pairs $\{\langle x, x\rangle \mid x \in M\}$ and, vice-versa, to obtain a strict ordering from an ordering by removing the pairs $\{\langle x, x\rangle \mid x \in M\}$.

In the sequel, we will be mostly concerned with orderings and ordered sets.

An <u>ordered set</u>, or <u>partially ordered set</u> is a pair $\langle M, \prec \rangle$, where M is a set and $\prec$ is an ordering on M. When no confusion can arise, we simply denote an ordered set $\langle M, \prec \rangle$ by M.

An <u>ordering</u> on a set M is <u>total</u> if:

$$\forall x,y \in M,\ x \prec y \text{ or } y \prec x \text{ holds.}$$

Let $\langle M, \prec \rangle$ be an ordered set, and $K \subseteq M$ be a subset of M. The restriction of $\prec$ to K is also an ordering on K. K is a <u>chain</u> in $\langle M, \prec \rangle$ when this ordering is total.

Let $\langle M, \prec \rangle$ be an ordered set and $A \subseteq M$ be a subset of M. We say that an element $u \in M$ such that:

$$\forall a \in A,\ a \prec u,$$

is an <u>upper bound</u> of A in M. If, in addition, $u \in A$, then u is the <u>greatest element</u> of A. There can obviously be only one greatest element of a set.

Similarly, if there is an element $\ell \in M$ such that:

$$\forall a \in A,\ \ell \prec a,$$

then ℓ is a lower bound of A in M. If, in addition, $\ell \in A$, then ℓ is the least element of A.

When the set of upper bounds of A in M has a least element we say that it is the least upper bound of A in M, and we denote it lub(A). Similarly, when the set of lower bounds of A in M has a greatest element, we say that it is the greatest lower bound of A in M, and we denote it glb(A).

When an ordered set $< M, \preccurlyeq >$ is such that every chain in M has a least upper bound, we say that it is chain-closed.

B.2 Monotonicity, Continuity.

Let f be a mapping over M (i.e. a mapping of M into M). For every subset $A \subseteq M$, f(A) denotes the set $\{f(x) \mid x \in A\}$.

An element $m \in M$ is a fixpoint of f if it such that $m = f(m)$. When the set of fixpoints of f has a least element, we say this least element is a least fixpoint of f (necessarily unique).

Let $< M, \preccurlyeq >$ be an ordered set. A mapping f over M is monotonic when:

$$\forall x, y \in M,\ x \preccurlyeq y \Rightarrow f(x) \preccurlyeq f(y) .$$

Let $< M, \preccurlyeq >$ be an ordered set which is chain closed. A mapping f over M is continuous when, for every chain $K \subseteq M$,

$$f(\text{lub}(K)) = \text{lub}(f(K)) \quad ,$$

where this is interpreted as meaning that the right hand side must exist and be equal to the left hand side.

As we noted in Chapter 4, continuity $\Rightarrow$ monotonicity.

B.3 Ordinals.

In order to prove the first Theorem in the next section, we need a few elementary properties of ordinals, which we briefly recall here, referring the reader to standard texts for a complete treatment. (See for example: Halmos (1960), Suppes (1960)).

We denote $<$ the (strict) total order relation on the ordinals.

An ordinal α is the set of its predecessors in that relation. An ordinal can be either 0, or the successor of another ordinal α (denoted $\alpha + 1$; in that case, every ordinal smaller than $\alpha + 1$ is either α or smaller than α), or a limit ordinal.

Let $\varphi(\alpha)$ denote a property of an ordinal α. If there is an ordinal α such that $\varphi(\alpha)$, then there exists a least ordinal β such that $\varphi\ \beta$). (Least Number Principle)

The following induction principle, called transfinite induction is valid on the ordinals:

If for every ordinal α, $\varphi(\beta)$ true for every $\beta < \alpha$ implies $\varphi(\alpha)$, then $\varphi(\alpha)$ holds for every ordinal α.

Cardinals are representatives of equivalence classes among ordinals, denoting their size. To every set A there is a cardinal number associated, denoted $|\ A\ |$, which is an ordinal equivalent to A, where equivalence here means having the same size. Given any cardinal, say $|\ A\ |$ for some set A, there is another cardinal strictly greater than it, namely $2^{|A|} = |\ \mathcal{P}(A)\ |$, which is the cardinal of the power set of A.

B.4 Least Upper Bounds

In this section, we consider partially ordered, chain-closed sets with a least element .

B.4.1 Monotonic Mappings.

We establish that every monotonic mapping over such a set has a least fixpoint, and give a characterization of that fixpoint when the mapping is continuous. The partially ordered set will be denoted $< M, \leq >$, or M for short, and its minimal element Ω. Let f be a monotonic mapping over M.

In the first paragraph, we show how to generalize the familiar Kleene sequence Ω, $f(\Omega)$, ..., $f^n(\Omega)$,... and define $f^\alpha(\Omega)$ for any ordinal α.

In the second paragraph we select one particular $f^\alpha(\Omega)$ in a suitable way, and call it s_f.

In the third paragraph, we show that s_f is the least fixpoint of f.

Definition of $f^\epsilon(\Omega)$ for every ordinal ϵ

We give an inductive definition of $f^\epsilon(\Omega)$ by:

Definition:

(i) if $\epsilon = 0$ then $f^\epsilon(\Omega) = \Omega$;

(ii) if $\epsilon = \delta + 1$ then $f^\epsilon(\Omega) = f(f^\delta(\Omega))$;

(iii) if ϵ is a limit ordinal, then $f^\epsilon(\Omega) = \text{lub}\{f^\alpha(\Omega) \mid \alpha < \epsilon\}$.

We will give here a justification that this definition indeed uniquely defines $f^\epsilon(\Omega)$ as an element of M for any ordinal ϵ, and

that this mapping is monotonic, in the sense that $\beta < \gamma \Rightarrow f^{\beta}(\Omega) \prec f^{\gamma}(\Omega)$.

Justification: We prove that, for every ordinal ϵ, the property $\varphi(\epsilon)$ holds, where:

$\varphi(\epsilon) \equiv$ " $f^{\epsilon}(\Omega)$ is well defined and, $\forall \beta < \epsilon$, $f^{\beta}(\Omega) \prec f^{\epsilon}(\Omega)$" .

The proof is by transfinite induction, i.e., we assume $\varphi(\alpha)$ for every $\alpha < \epsilon$ and prove $\varphi(\epsilon)$.

Three cases arise:

(i) $\epsilon = 0$: $f^{0}(\Omega) = \Omega$ is well defined, and there is no ordinal < 0, so the second half of $\varphi(0)$ is vacuously true.

(ii) $\epsilon = \delta + 1$.

Then $f^{\epsilon}(\Omega) = f(f^{\delta}(\Omega))$ is a well defined element of M, since $f^{\delta}(\Omega)$ is a well defined element of M and f is a mapping over M.

Now let $\beta < \epsilon$, show $f^{\beta}(\Omega) \prec f^{\epsilon}(\Omega)$. Since δ is the predecessor of ϵ, we have $\beta \leq \delta$, and we know that $\varphi(\delta)$ holds.

We can distinguish three cases: $\beta = 0$, $\beta = \delta \neq 0$, $\beta < \delta$

(a) $\beta = 0$

Then $f^{\beta}(\Omega) = \Omega$ and, since Ω is the least element of M, $\Omega \prec f^{\epsilon}(\Omega)$ trivially.

(b) $\beta = \delta \neq 0$.

Then δ has a predecessor γ, and we have:

$$f^{\gamma}(\Omega) \prec f^{\delta}(\Omega), \text{ by } \varphi(\delta) .$$

By monotonicity of f, we have:

$$f(f^{\gamma}(\Omega) \prec f(f^{\delta}(\Omega)) ,$$

i.e. $f^{\gamma+1}(\Omega) \prec f^{\delta+1}(\Omega)$,

i.e. $f^{\beta}(\Omega) \prec f^{\epsilon}(\Omega)$, since $\beta = \delta = \gamma + 1$ and $\epsilon = \delta + 1$.

(c) $\beta < \delta$

In this case, we know $f^{\beta}(\Omega) \prec f^{\delta}(\Omega)$ by $\varphi(\delta)$, and we have established $f^{\delta}(\Omega) \prec f^{\epsilon}(\Omega)$ in cases (a) or (b).

So the property follows by transitivity of $\prec$.

(iii) ϵ is a limit ordinal.

Then $f^{\epsilon}(\Omega) = \text{lub}\{f^{\alpha}(\Omega) \mid \alpha < \epsilon\}$.

We want to show that $K_{\epsilon} = \{f^{\alpha}(\Omega) \mid \alpha < \epsilon\}$ is a chain. Let $\alpha, \beta < \epsilon$. Since the ordinals are totally ordered, we have either $\alpha < \beta$ or $\beta < \alpha$ or $\alpha = \beta$. By induction hypothesis we know that $\varphi(\alpha)$ and $\varphi(\beta)$ hold. Therefore, if $\alpha < \beta$, we have $f^{\alpha}(\Omega) \prec f^{\beta}(\Omega)$; if $\beta < \alpha$, $f^{\beta}(\Omega) \prec f^{\alpha}(\Omega)$, and of course, if $\beta = \alpha$, $f^{\alpha}(\Omega) = f^{\beta}(\Omega)$, because of well-definedness, hence $f^{\alpha}(\Omega) \prec f^{\beta}(\Omega)$ because of reflexivity. Hence the order is total on K_{ϵ}, and K_{ϵ} is a chain. Since M is chain closed, $\text{lub}(K_{\epsilon})$ is a well defined element of M, and so is $f^{\epsilon}(\Omega)$.

Now the second part of $\varphi(\epsilon)$ holds trivially, because if $\beta < \epsilon$ then $f^{\beta}(\Omega) \in K_{\epsilon}$ and therefore:

$f^{\beta}(\Omega) \prec f^{\epsilon}(\Omega)$, since $f^{\epsilon}(\Omega)$ is an upper bound of K_{ϵ}.

So $\varphi(\epsilon)$ holds for every ordinal ϵ, which justifies the definition of $f^{\epsilon}(\Omega)$ and proves that:

for every ordinal β, γ: $\beta < \gamma \Rightarrow f^{\beta}(\Omega) \prec f^{\gamma}(\Omega)$.

□

Construction of s_f.

The key point is that the $f^{\epsilon}(\Omega)$ that we have defined in the previous paragraph cannot be all distinct. Indeed, since $f^{\epsilon}(\Omega) \in M$ for every ordinal ϵ, there cannot be more distinct $f^{\epsilon}(\Omega)$ than there are distinct elements in M. If we take ϵ sufficiently large so that its cardinal is greater than $|M|$, then there must be two ordinals $\alpha,\beta \leq \epsilon$ such that $f^{\alpha}(\Omega) = f^{\beta}(\Omega)$. It is enough for this to take an ordinal equivalent to $\mathcal{P}(M)$, where $\mathcal{P}(M)$ denotes the power set of M. For the purposes of this discussion, let us call conjugates any two distinct ordinals α,β such that $f^{\alpha}(\Omega) = f^{\beta}(\Omega)$, and let $P(\alpha)$ be the property "α has a conjugate". We know that $P(\epsilon)$ is true for the ϵ that we have taken above, and therefore, by the least number principle, there must be a least ordinal that has a conjugate. Let us denote it ϵ_f.

Then we set $s_f = f^{\epsilon_f}(\Omega)$ and this completes our construction of s_f.

Theorem.

If M is a partially ordered, chain-closed set, which has a least element, then every monotonic mapping over M has a least fixpoint.

Proof. Let f be a monotonic mapping over M. We prove that s_f, as defined in the previous paragraph, is a least fixpoint of f.

(a) s_f is a fixpoint of f.

We have $s_f = f^{\epsilon_f}(\Omega)$. We want to prove $s_f = f(s_f)$, i.e., $f^{\epsilon_f}(\Omega) = f^{\epsilon_f+1}(\Omega)$. Let λ be a conjugate of ϵ_f.

We have $f^{\epsilon_f}(\Omega) = f^{\lambda}(\Omega)$ and $\epsilon_f < \lambda$.

If $\lambda = \epsilon_f + 1$ we are done.

Assume $\lambda \neq \epsilon_f + 1$. Then $\epsilon_f + 1 < \lambda$. Therefore, we have:

$$f^{\epsilon_f}(\Omega) \prec f^{\epsilon_f + 1}(\Omega) \prec f^{\lambda}(\Omega) .$$

But $f^{\lambda}(\Omega) = f^{\epsilon_f}(\Omega)$, so, by antisymmetry of $\prec$, we get:

$$f^{\epsilon_f}(\Omega) = f^{\epsilon_f + 1}(\Omega).$$

(b) s_f is the least fixpoint of f.

Let m be a fixpoint of f. We want to show $s_f \prec m$.

To prove this, we will show that, for every ordinal ϵ, $f^{\epsilon}(\Omega) \prec m$. The proof is by structural induction.

(i) $\epsilon = 0$. Trivial, since $\Omega \prec m$ for any $m \in M$.

(ii) $\epsilon = \delta + 1$. By induction hypothesis $f^{\delta}(\Omega) \prec m$.

By monotonicity $f(f^{\delta}(\Omega)) \prec f(m)$.

Therefore $f^{\epsilon}(\Omega) \prec m$, since $\epsilon = \delta + 1$ and m is a fixpoint of f.

(iii) ϵ is a limit ordinal. By induction hypothesis, $\forall \alpha < \epsilon$, $f^{\alpha}(\Omega) \prec m$. By definition of $f^{\epsilon}(\Omega)$, we have $f^{\epsilon}(\Omega) = \text{lub} K_{\epsilon}$, where $K_{\epsilon} = \{f^{\alpha}(\Omega) \mid \alpha < \epsilon\}$.

By induction hypothesis, m is an upper bound of K_{ϵ}, and therefore $f^{\epsilon}(\Omega) \prec m$.

In particular, if we take $\epsilon = s_f$, we obtain:

$$s_f = f^{s_f}(\Omega) \prec m.$$

□

B.4.2 Continuous Mappings.

If the monotonic mapping f that we considered

in paragraph B.4.1 is now continuous, the least fixpoint has a much simpler characterization than before. In fact we have:

Theorem:

If M is a partially ordered, chain closed set which has a least element Ω, and if f is a continuous mapping over M, then the least fixpoint of m is $\text{lub}\{f^i(\Omega) \mid i \text{ nonnegative integer}\}$.

Proof: The results of the previous section apply since if f is continuous it is monotonic.

We have shown that, for every fixpoint m of f and every ordinal α, $f^\alpha(\Omega) \preceq m$.

In particular, if s_f is the least fixpoint of f, for every non negative integer i, $f^i(\Omega) \preceq s_f$.

The first infinite ordinal is ω, where $\omega = \{i \mid i \text{ non negative integer}\}$

$$= \{i \mid 0 \leq i < \omega\},$$

and we have also $f^\omega(\Omega) \preceq s_f$, where, by definition of $f^\omega(\Omega)$:

$f^\omega(\Omega) = \text{lub}\{f^i(\Omega) \mid 0 \leq i < \omega\}$.

Let $K = \{f^i(\Omega) \mid 0 \leq i < \omega\}$. By continuity of f, $f(\text{lub}(K)) = \text{lub}(f(K))$.

We have:

$$\text{lub}(f(K)) = \text{lub}\{f^{i+1}(\Omega) \mid i + 1\}.$$

But $f(K) = \{f^i(\Omega) \mid 0 < i < \omega\}$,

and therefore $\text{lub}(f(K)) = \text{lub}(K)$.

Hence : $\text{lub}(K) = f(\text{lub}(K))$,

which means that $\text{lub}(K) = f^\omega(\Omega)$ is a fixpoint of f.

Hence $s_f \preceq f^\omega(\Omega)$.

Therefore $s_f = f^\omega(\Omega) = \text{lub}\{f^i(\Omega) \mid i \text{ non negative integer}\}$. □

B.5 Greatest Lower Bounds

If the partially ordered set M now has the property that every nonempty subset of M has a glb in M, then monotonic mappings over M turn out to have another interesting property:

Theorem:

If M is a partially ordered set in which every nonempty subset has a glb in M, then every monotonic mapping f over M which has a fixpoint has a least fixpoint s_f which is such that:

$$s_f = \text{glb}\ \{m \in M \mid f(m) \prec m\}\ .$$

Proof: This proof essentially follows Park (1969) in his proof of the Knaster-Tarski theorem. Since f has a fixpoint, the set $C_f = \{m \in M \mid f(m) \prec m\}$ is not empty, and has a glb. Let us temporarily denote $l_f = \text{glb}(C_f)$.

(a) l_f is a fixpoint of f:

For every $l \in C_f$, we have $l_f \prec l$ and, by monotonicity of f, $f(l_f) \prec f(l)$. But $f(l) \prec l$ since $l \in C_f$, and therefore $f(l_f) \prec l$. Hence $f(l_f)$ is a lower bound of C_f, therefore:

$$f(l_f) \prec l_f. \qquad (1)$$

By monotonicity of f, we get from (1) that $f(f(l_f)) \prec f(l_f)$, which means that $f(l_f) \in C_f$.

Hence: $$l_f \prec f(l_f)\ . \qquad (2)$$

(1) and (2) imply that l_f is a fixpoint of f.

(b) l_f is the least fixpoint of f:

We know that f has a fixpoint, so the set of its fixpoints is not empty and therefore has a glb which we call temporarily m_f. (We do not know yet that m_f is a fixpoint of f).

Since the set of fixpoints of f is included in C_f, the glb's of these sets are in the following relation: $l_f \preccurlyeq m_f$.

But, by (a) above l_f is a fixpoint of f, hence $m_f \preccurlyeq l_f$.

Therefore we have: $l_f = m_f$. □

A sufficient condition on M for a monotonic f to have a fixpoint was given in B.3, namely that M be chain-closed and have a least element.

This previous theorem provides an elegant proof for the following corollary:

Corollary Let M verify the hypothesis of B.3, and let f_1, f_2 be two monotonic mappings on M with fixpoints. Call s_1 and s_2 their least fixpoints. Then if $(\forall m \in M)\ [f_1(m) \preccurlyeq f_2(m)]$ then $s_1 \preccurlyeq s_2$.

Proof. From the hypothesis, we have in particular:

$$f_1(s_2) \preccurlyeq f_2(s_2).$$

But $f_2(s_2) = s_2$. Hence $f_1(s_2) \preccurlyeq s_2$.

Hence $s_2 \in C_{f_1}$, and, therefore $s_1 \preccurlyeq s_2$ since $s_1 = \text{glb}(C_{f_1})$. □

C. Partial Functions and the Extension Relation

In this section we show how the set of partial functions from a domain S into a range R can be partially ordered by the extension relation, and we apply the results of Section B to this structure.

C.1 Definitions

Let S, R be non-empty sets and ω be an element not in R, representing the "undefined" element. We denote $R \cup \{\omega\}$ by R^+. A partial function of S into R is a mapping of S into R^+. We denote $pf(S \to R)$ the set of partial functions of S into R. We abbreviate $pf(S \to S)$ by $pf(S)$ and we call a partial function of S into S a partial function over S.

If $f \in pf(S \to R)$, we say that S is the domain of f; for every $x \in S$, the image of x by f is denoted $f(x)$; we denote $=$ the equality relation on R^+; if $f(x) = \omega$ then f is said to be undefined at x, (or $f(x)$ is said to be undefined); if $f(x) \neq \omega$, then f is said to be defined at x (or $f(x)$ defined); if f is defined for every $x \in S$, then f is said to be total; if f is undefined for every $x \in S$, then f is the undefined function, denoted Ω.

Let $f, g \in pf(S \to R)$. Then:

(a) f and g are said to be equal, denoted $f = g$, iff, $\forall x \in S$, $f(x) = g(x)$. [i.e., either both $f(x)$ and $g(x)$ are undefined, or they are both defined and equal as elements of R].

(b) g is said to be an <u>extension</u> of f, denoted $f \prec g$, iff, $\forall x \in S$, $f(x) \neq \omega \Rightarrow f(x) \equiv g(x)$. (i.e., wherever f(x) is defined, g(x) must be equal to it, and therefore also defined). Note that $f = g$ iff $f \prec g$ and $g \prec f$.

[An equivalent way of defining the extension relation is to introduce a partial ordering $\prec$ on R^+ by:

$$\forall y \in R^+, \quad \omega \prec y \quad \text{and} \quad y \prec y,$$

and then to say that:

$\forall f,g \in pf(S \rightarrow R)$, $f \prec g$ (g is an extension of f) $\Leftrightarrow \forall x \in S, f(x) \prec g(x)]$.

The extension relation is easily seen to be:

(i) Reflexive (i.e. $\forall f \in pf(S \rightarrow R)$, $f \prec f$);

(ii) Antisymmetric (i.e. $\forall f,g \in pf(S \rightarrow R)$, $f \prec g$ and $g \prec f \Rightarrow f = g$);

(iii) Transitive (i.e. $\forall f,g,h \in pf(S \rightarrow R)$, $f \prec g$ and $g \prec h \Rightarrow f \prec h$).

Therefore, $\prec$ is a partial ordering on $pf(S \rightarrow R)$.

We are now going to study the properties of $pf(S \rightarrow R)$ partially ordered by $\prec$.

C.2 <u>Properties of sets of Partial Functions</u>

<u>$pf(S \rightarrow R)$ has a least element</u>

It is immediate that Ω, the totally undefined partial function, is a least element in $pf(S \rightarrow R)$, i.e., that :

$\forall f \in pf(S \rightarrow R)$, $\Omega \prec f$.

pf(S → R) is chain-closed:

Proof: Let $K \subset pf(S \to R)$ be a chain. Let $f_1, f_2 \in K$, and let $x \in S$ be such that f_1 and f_2 are both defined at x. Since $\prec$ is a total ordering on K, we must have either $f_1 \prec f_2$ or $f_2 \prec f_1$. In either case, $f_1(x) = f_2(x)$, by definition of $\prec$.

Using this property, we can define a partial function U_K in $pf(S \to R)$ as follows:

$$\forall x \in S \; : \; U_K(x) = \begin{cases} \text{if } \exists f \in K \text{ such that } f(x) \neq \omega \text{ then } f(x) \\ \text{otherwise } \perp \; . \end{cases}$$

(Because of the opening remark, if there are several f's in K such that $f(x) \neq \omega$, it does not matter which one we pick to define $U_K(x)$ since they have all the same value at x).

We want to show : $U_K = \text{lub}(K)$.

(a) U_K is an upper bound of K.

Let $f \in K$ and $x \in S$. Assume $f(x) \neq \omega$. Then, by definition of U_K, $U_K(x) = f(x)$. Therefore:

$$\forall f \in K, \quad f \prec U_K \; .$$

(b) U_K is the least upper bound of K.

Assume g is an upper bound of K. Let $x \in S$ and assume $U_K(x) \neq \omega$. Then, by definition of U_K, there exists $f \in K$ such that $f(x) \neq \omega$ and $U_K(x) = f(x)$. Since $f \prec g$, we also have $f(x) = g(x)$. Therefore $U_K \prec g$.

Every non-empty subset of $pf(S \to R)$ has a glb[*]

Proof: Let $M \subseteq pf(S \to R)$, $M \neq \emptyset$.

Let us define $\ell_M(x) \in pf(S \to R)$ by:

$$\ell_M(x) = \begin{cases} \text{if } \exists z \in R \text{ such that } (\forall f \in M)(f(x) = z) \text{ then } z \\ \text{otherwise } \omega . \end{cases}$$

ℓ_M is a partial function from S into R, where $M \neq \emptyset$, because the z of the definition is then unique.

We show that $\ell_M = glb(M)$:

(a) ℓ_M is a lower bound of M.

From the definition of ℓ_M it follows that, if $\ell_M(x)$ is defined, then $\forall f \in M$, $\ell_M(x) = f(x)$. This implies that $\forall f \in M$, $\ell_M \prec f$.

(b) ℓ_M is the greatest lower bound of M.

Let ℓ be a lower bound of M. Let $x \in S$ and assume $\ell(x) \neq \omega$. For every $f \in M$, we have $\ell \prec f$, and so $f(x) = \ell(x)$. Therefore there exists a z as required in the definition of $\ell_M(x)$, namely $z = \ell(x)$. Thus $\ell_M(x) = \ell(x)$. This implies $\ell \prec \ell_M$ for every lower bound ℓ of M.

□

Therefore we see that $pf(S \to R)$ structured by the extension relation verifies all the properties that we need in order to apply the fixpoint theorems of Section B.4. and B.5. We will call a mapping over $pf(S \to R)$ a functional. Combining the three results,

---

[*] For $|R| > 1$, the empty subset of $pf(S \to R)$ does not have a greatest lower bound in $pf(S \to R)$, because there is no partial function in $pf(S \to R)$ that is an extension of all the others. In fact, this is the only property that $pf(S \to R)$ lacks in order to be a complete lattice, (since it is sufficient, for an ordered set to be a complete lattice, that every subset of the set possess a glb). Scott (1970) structures $pf(S \to R)$ into a complete lattice by adding a "top" element, which he denotes $\top$, extending $\prec$ by $\forall f \in pf(S \to R) \cup \{R\}$, $f \prec \top$. Then $\top = glb(\emptyset)$. But this is unnecessary for our discussion in this work.

we can state:

Theorem: Every monotonic functional τ over $pf(S \to R)$ has a least fixpoint s_τ which is equal to $glb\{f \in pf(S \to R) \mid \tau(f) \prec f\}$.

If, in addition, τ is continuous, then:

$$s_\tau = lub\ \{\tau^i(\Omega) \mid i \text{ non negative integer}\}$$

C.3. Properties of Sets of Monotonic Partial Functions

If we have a partial ordering on S, which we denote $\prec$, and if we take the natural partial ordering $\prec$ on R^+ defined by: $\forall\ y \in R^+$, $\omega \prec y$ and $y \prec y$, we have the natural definition of monotonicity, namely $f \in pf(S \to R)$ is monotonic when:

$$\forall\ x,y \in S,\quad x \prec y \Rightarrow f(x) \prec f(y).$$

Let us denote $mf(S \to R)$ the set of monotonic partial functions of S into R.

We are going to show that $mf(S \to R)$ also satisfies the basic properties needed to apply the general theorems of part B.

$mf(S \to R)$ has a least element

Ω is obviously monotonic, and is therefore the least element of $mf(S \to R)$.

$mf(S \to R)$ is chain-closed

We will show that if K is a chain of monotonic partial functions,

the lub U_K of K in $pf(S \to R)$ is monotonic, which implies that $mf(S \to R)$ is chain-closed.

We have seen in section C.2 that:

$$\forall x \in S : U_K(x) = \begin{cases} \text{If } \exists f \in K \text{ such that } f(x) \neq \omega \text{ then } f(x) \\ \text{otherwise } \omega . \end{cases}$$

Let $x, y \in S$ such that $x \prec y$. Assume $U_K(x)$ is defined. Then there is an f in K such that $f(x) \equiv U_K(x) \neq \omega$. Because $K \subseteq mf(S \to R)$, f is monotonic and therefore $f(x) \equiv f(y) \neq \omega$. Hence, applying again the definition of U_K, we see that $U_K(y) = f(y) \neq \omega$.

Therefore we have $U_K(x) \neq \omega \Rightarrow U_K(x) \equiv U_K(y)$,

i.e. $U_K(x) \prec U_K(y)$, which shows that $U_K \in mf(S \to R)$.

□

Every non-empty subset of $mf(S \to R)$ has a glb in $mf(S \to R)$

We show that $l_M(x) = \begin{cases} \text{if } \exists z \in R \text{ such that } (\forall f \in M)\ (f(x)=z) \text{ then } z \\ \text{otherwise } \perp \end{cases}$

is monotonic.

Let $x, y \in S$ such that $x \prec y$. Assume $l_M(x) \neq \omega$.

Then $\forall f \in M$, $f(x) \equiv l_M(x) \neq \omega$. Since every f in M is monotonic, we also have:

$$\forall f \in M , f(x) \equiv f(y) \neq \omega .$$

Hence, by definition of l_M, $l_M(y) \equiv l_M(x)$, which shows monotonicity of l_M.

□

Therefore $mf(S \to R)$ satisfies the hypothesis of the theorems of Part B, and, combining the results, we can state:

Theorem: Every monotonic functional τ over $mf(S \to R)$ has a monotonic least fixpoint s_τ which is equal to:

$$glb \{ f \in mf(S \to R) \mid \tau(f) \leq f \} .$$

If, in addition, τ is continuous, then:

$$s_\tau = lub \{ \tau^i (\Omega) \mid i \text{ non negative integer} \} .$$

APPENDIX II

SYSTEMS OF RECURSIVE DEFINITIONS

A. Introduction

B. Extension of the Model of Chapter 2

C. Analogs of the Results of Chapter 3

D. Analogs of the Results of Chapter 4

E. Fixpoint Computations for Systems of Recursive Definitions

A. Introduction

In this Appendix, we extend the model and the results of Chapters 2 - 5 to systems of recursive definitions. We show how all definitions can be extended in a natural way so that the results valid for single recursive definitions possess analogs which are valid for systems of recursive definitions; these reduce to the previous ones when the system reduces to a single definition. The extensions are rather straightforward except for the notations which unfortunately tend to obscure the simplicity of the results.

B. Extension of the Model of Chapter 2

In order to express systems of recursive definitions, we need a straightforward extension of the alphabet and terms of Chapter 2. The alphabet is extended to include k function variable letters, F_1, F_2, ..., F_k, and the terms are constructed in the same way as in § 2.2.2 except that 2.2.2 (v) is changed to: "If τ_1, ..., τ_n are terms, then $F_j(\tau_1, \ldots, \tau_n)$ is a term, for every j, $1 \leq j \leq k$."

Such terms are now denoted by $\sigma(\bar{F},\bar{x})$, $\rho(\bar{F},\bar{x})$, ..., $\sigma(\bar{F},\bar{x})$, $\tau(\bar{F},\bar{x})$, etc...

The definition of substitution in § 2.2.4 is trivially generalized for these terms. The definition of interpretations in § 2.3.1 need not be modified.

The weak and strong values of terms are easily generalized in the following way:

(a) Weak Values

Given a k-tuple $\bar{f} = \langle f_1, \ldots, f_k \rangle$ of partial functions

$f_j \in pf_n(D)$, $1 \le j \le k$, and an n-tuple $\bar{\xi} = \langle \xi_1, \ldots, \xi_n \rangle$ of elements $\xi_i \in \Delta^+ \cup \{d\}$, the weak value of a term α for $\bar{F} = \bar{f}$ and $\bar{x} = \bar{\xi}$, denoted $\tilde{\alpha}(\bar{f},\bar{\xi})$, is defined inductively as in § 2.3.2 (a), except for case (v) which is modified as follows:

"if $\alpha = F_j(\alpha_1, \ldots, \alpha_n)$ for some j, $1 \le j \le k$, then:

- if the vector $\langle \tilde{\alpha}_1(\bar{f},\bar{\xi}), \ldots, \tilde{\alpha}_n(\bar{f},\bar{\xi}) \rangle$ does not belong to $(D^+)^n$, then $\tilde{\alpha}(\bar{f},\bar{\xi}) \equiv d$,
- otherwise, if, for some i, $1 \le i \le n$, $\tilde{\alpha}_i(\bar{f},\bar{\xi}) \equiv \omega$, then $\tilde{\alpha}(\bar{f},\bar{\xi}) \equiv \omega$,
- otherwise, $\tilde{\alpha}(\bar{f},\bar{\xi}) \equiv f_j(\tilde{\alpha}_1(\bar{f},\bar{\xi}), \ldots, \tilde{\alpha}_n(\bar{f},\bar{\xi}))$."

(b) Strong Values

Given a k-tuple $\bar{f} = \langle f_1, \ldots, f_k \rangle$ of partial functions $f_j \in pf_n(D^+)$, $1 \le j \le k$ and an n-tuple $\bar{\xi} = \langle \xi_1, \ldots, \xi_n \rangle$ of elements $\xi_i \in \Delta \cup \{d\}$, $1 \le i \le n$, the strong value of a term α for $\bar{F} = \bar{f}$ and $\bar{x} = \bar{\xi}$, denoted $\tilde{\tilde{\alpha}}(\bar{f},\bar{\xi})$ is defined inductively as in § 2.3.2. (b), except for case (v) which is modified as follows:

"if $\alpha = F_j(\alpha_1, \ldots, \alpha_n)$ for some j, $1 \le j \le k$, then:

- if the vector $\langle \tilde{\tilde{\alpha}}_1(\bar{f},\bar{\xi}), \ldots, \tilde{\tilde{\alpha}}_n(\bar{f},\bar{\xi}) \rangle$ does not belong to $(D^+)^n$, then $\tilde{\tilde{\alpha}}(\bar{f},\bar{\xi}) \equiv d$,
- otherwise: $\tilde{\tilde{\alpha}}(\bar{f},\bar{\xi}) \equiv f_j(\tilde{\tilde{\alpha}}_1(\bar{f},\bar{\xi}), \ldots, \tilde{\tilde{\alpha}}_n(\bar{f},\bar{\xi}))$."

Comments similar to those in § 2.3.5 may be made regarding the relations between the two types of evaluations. In particular, for every k-tuple $\bar{f} = \langle f_2, \ldots, f_k \rangle$ of partial functions $f_j \in pf_n(D)$, $1 \le j \le k$,

and for every $\bar{f} \in D^n$, we have, for every term $\alpha(\bar{F}, \bar{x})$:

$$\tilde{\alpha}(\bar{f}, \bar{x}) \equiv \tilde{\alpha}(\bar{f}^+, \bar{x}),$$

where $\bar{f}^+$, natural extension of $\bar{f}$ is defined by:

$$\bar{f}^+ = \langle f_1^+, \ldots, f_k^+ \rangle .$$

Correct terms are defined as terms whose strong values always belong to Δ^+, and compatible terms as terms whose strong values always belong to D^+; the formal definitions are straightforward extensions of those in § 2.3.2.

A system of recursive definitions will be a system of the form:

$$\begin{cases} F_1(\bar{x}) \Leftarrow \tau_1(\bar{F}, \bar{x}) \\ \quad \vdots \\ F_k(\bar{x}) \Leftarrow \tau_k(\bar{F}, \bar{x}) \end{cases},$$

where there is exactly one recursive definition for each F_j, $1 \leq j \leq k$, and where each τ_j, $1 \leq j \leq k$, is a compatible term.

We abbreviate such a system by the notation:

$$\bar{F}(\bar{x}) \Leftarrow \bar{\tau}(\bar{F}, \bar{x}).$$

Fixpoints of such systems are defined in the following way, which is a natural generalization of the definitions of § 2.3.4:

(a) Weak Fixpoints:

A k-tuple $\bar{f} = \langle f_1, \ldots, f_k \rangle$ of partial functions $f_j \in pf_n(D)$, $1 \leq j \leq k$, is a weak fixpoint of a system of

recursive definitions $\bar{F}(\bar{x}) \Leftarrow \bar{\tau}(\bar{F}, \bar{x})$ iff:

$$\forall \bar{\xi} \in D^n, \forall j, 1 \le j \le k: \quad f_j(\bar{\xi}) \equiv \tilde{\tau}_j(\bar{f}, \bar{\xi})$$

(b) Strong Fixpoints

A k-tuple $\bar{f} = \langle f_1, \ldots, f_k \rangle$ of partial functions $f_j \in pf_n(D^+)$, $1 \le j \le k$, is a strong fixpoint of a system of recursive definitions, $\bar{F}(\bar{x}) \Leftarrow \bar{\tau}(\bar{F}, \bar{x})$,

iff:

$$\forall \bar{\xi} \in (D^+)^n, \forall j, 1 \le j \le k, \; f_j(\bar{\xi}) \equiv \tilde{\tilde{\tau}}_j(\bar{f}, \bar{\xi}) .$$

It is convenient to associate functionals to a system of recursive definitions, or, more generally, to a k-tuple of correct terms. Let us show, as an example, how to associate a functional $\tilde{\tilde{\alpha}}$ to a k-tuple $\bar{\alpha} = \langle \alpha_1, \ldots, \alpha_k \rangle$ of correct terms.

$\tilde{\tilde{\alpha}}$ will be a functional mapping an element $\bar{f}$ of $(D^+)^n \to (D^+)^k$ */ into an element $\tilde{\bar{\alpha}}[\bar{f}]$ defined as follows :

Let $\bar{f} = \langle f_1, \ldots, f_k \rangle$ be an element of $(D^+)^n \to (D^+)^k$, where, for every j, $1 \le j \le k$, f_j is an element of $(D^+)^n \to D^+ = pf_n(D^+)$; then $\tilde{\tilde{\alpha}}[\bar{f}]$ maps every $\bar{\xi} \in (D^+)^n$ into $\tilde{\bar{\alpha}}[\bar{f}](\bar{\xi}) \in (\Delta^+)^k$ defined by:

$$\tilde{\bar{\alpha}}[\bar{f}](\bar{\xi}) = \langle \tilde{\tilde{\alpha}}_1(\bar{f}, \bar{\xi}), \ldots, \tilde{\tilde{\alpha}}_k(\bar{f}, \bar{\xi}) \rangle ,$$

where, for every j, $1 \le j \le k$, $\tilde{\tilde{\alpha}}_j(\bar{f}, \bar{\xi})$ is the strong value of α_j for $\bar{F} = \bar{f}$ and $\bar{x} = \bar{\xi}$.

---

*/ We use the notation '$S \to R$' to denote the set of mappings of S into R. Notice that $(D^+)^n \to (D^+)^k$ is trivially isomorphic to $[pf_n(D^+)]^k$.

If $\bar{\alpha}$ is a k-tuple of compatible terms, then $\tilde{\tilde{\bar{\alpha}}}$ is simply a functional over $(D^+)^n \to (D^+)^k$. If we take a system of recursive definitions $\bar{F}(\bar{x}) \Leftarrow \bar{\tau}(\bar{F},\bar{x})$, in which $\bar{\tau} = \langle \tau_1, \ldots, \tau_k \rangle$, then $\bar{\tau}$ is a k-tuple of compatible terms, and it is clear that a strong fixpoint of the system, as we have defined it above, is equivalently a fixpoint of the functional $\tilde{\tilde{\bar{\tau}}}$.

A completely analogous discussion can be carried out for the weak fixpoints.

We now need to generalize the definition of computations of § 2.4.2. to systems. An elementary computation of a term $\alpha(\bar{F},\bar{x})$ for $\bar{x} = \bar{c} \in (D^+)^n$, using a system of recursive definitions $\bar{F}(\bar{x}) \Leftarrow \bar{\tau}(\bar{F},\bar{x})$ is a sequence of terms $\{\alpha_i, i \geq 0\}$ such that:

(a) $\alpha_0 = \alpha(\bar{F},\underset{\sim}{c})$ */;

(b) For every $i \geq 0$:

- either (b1): $\alpha_{i+1} \in S^{\bar{\rho}}_{\gamma}\, \alpha_i$, where ρ is free of $\bar{x}$, and the equation:

$$\tilde{\tilde{\rho}}(\bar{f}, \;) = \tilde{\tilde{\gamma}}(\bar{f}, \;) \text{ holds for every } \bar{f} \in (D^+)^n \to (D^+)^k,$$

- or (b2):

$$\alpha_{i+1} \in S^{\tau_j(\bar{F},\bar{\rho})}_{F_j(\bar{\rho})}\, \alpha_i \text{ for some } j,\ 1 \leq j \leq k.$$

---

*/ The definition of $\alpha(\bar{F},\bar{\rho})$ where $\bar{\rho}$ is an n-tuple of terms is a straightforward adaptation of the definition given in § 2.4.1.

The definitions of computations, terminating computations and innermost computations are easily transposed from those in Chapter 2.

A computed function over $(D^+)^n$ of a system of recursive definitions $\bar{F}(\bar{x}) \Leftarrow \bar{\tau}(\bar{F},\bar{x})$ is any $\bar{f} \in (D^+)^n \to (D^+)^k$ such that, for every $\bar{c} \in (D^+)^n$ and every j, $1 \leq j \leq k$:

(i) if $f_j(\bar{c}) \neq \omega$[*], then there is a computation of $F_j(\bar{x})$ for $\bar{x} = \bar{c}$, using the system of recursive definitions, which terminates with $f_j(\bar{c})$;

(ii) if $f_j(\bar{c}) = \omega$, then there is a computation of $F_j(\bar{x})$ for $\bar{x} = \bar{c}$, using the system of recursive definitions, which does not terminate.

Similarly, one defines a computed function over D^n by substituting D^n for $(D^+)^n$ everywhere in the above definition.

C. Analogs of the Results of Chapter 3

With these definitions, all the results in Chapters 3 - 5 are easily translated into results for systems of recursive definitions.

---

[*] if $\bar{f} \in (D^+)^n \to (D^+)^k$, for every j, $1 \leq j \leq k$, f_j is the partial function of $pf_n(D^+)$ defined by: $\forall \bar{e} \in (D^+)^n : f_j(\bar{e})$ is the j-th component of $\bar{f}(\bar{e}) \in (D^+)^k$ [j-th projection]

For example we have:

(a) Theorem 1: For every system of recursive definitions, every strong fixpoint is an extension[1] of every computed function over $(D^+)^n$.

To prove this, one applies the same techniques as in Chapter 3 to each component $F_j(x)$. All the preparatory lemmas of § 3.2.1, § 3.2.2 and § 3.2.3 apply with the obvious change of notations: F into $\bar{F}$, $pf_n(D^+)$ into $(D^+)^n \to (D^+)^k$, etc.

(b) Similarly, the lemmas and Theorem 2 of Section 3.3 concerning innermost computed functions and weak fixpoints extend immediately to systems of recursive definitions.

D. Analogs of the Results of Chapter 4.

The study of monotonicity and continuity for systems of recursive definitions proceeds along the lines of Chapter 4, except that it is better to regard $(D^+)^n \to (D^+)^k$ as a partially ordered set with the appropriate properties and apply the general theorems of § 4.4.1 to it rather than to try to regard it as a set of partial functions.

The notion of a monotonically structured system of recursive definitions is the same as in the case of a single recursive definition: the given functions must be monotonic.

---

[1] If $\bar{f}$, $\bar{g}$ are elements of $(D^+)^n \to (D^+)^k$, we say that $\bar{g}$ is an extension of $\bar{f}$, denoted $\bar{f} \prec \bar{g}$, if and only if: $\forall j$, $1 \leq j \leq k$, $f_j \prec g_j$ (where f_j and g_j are the j-th projections of $\bar{f}$ and $\bar{g}$, which are elements of $pf_n(D^+)$).

Then, for monotonically structured terms, the lemmas of § 4.3.2 , 4.3.3 , 4.3.4 , and 4.3.5 concerning monotonicity and continuity carry over to the case of several function variables without any difficulty. The relevant partial orderings on $(D^+)^n \to (D^+)^k$ and $D^n \to (D^+)^k$ are <u>extension</u> orderings denoted $\preceq$ and defined in the following way:

$\forall\, \bar{f}, \bar{g} \in (D^+)^n \to (D^+)^k : \bar{f} \preceq \bar{g}$ iff $\forall_j$, $1 \le j \le k$, $f_j \preceq g_j$;

$\forall\, \bar{f}, \bar{g} \in D^n \to (D^+)^k : \bar{f} \preceq \bar{g}$ iff $\forall_j$, $1 \le j \le k$, $f_j \preceq g_j$.

In fact the functionals $\tilde{\approx}$ and $\tilde{=}$ associated with a monotonically structured system of recursive definitions also have properties of monotonicity and continuity, from which the analogs of Theorems 3 and 4 follow directly. Let us go into some detail for the derivation of the analog of Theorem 3:

Let $\bar{F}(\bar{x}) \Leftarrow \bar{=}(\bar{F}, \bar{x})$ be a monotonically structured system of recursive definitions. Then we first show that $\tilde{=}$ is a monotonic and continuous mapping over the set of monotonic mappings of $(D^+)^n \to (D^+)^k$ i.e. over $[mf_n(D^+)]^k$. The arguments follow those of § 4.3.2. and § 4.3.4., using in addition the fact that $\tilde{\approx}$ is monotonic (resp. continuous) iff for every j, $1 \le j \le k$, $\tilde{\approx}_j$ is monotonic (resp. continuous).

Then one shows that the set $M = [mf_n(D^+)]^k$ has a least element, $\bar{\Omega} = \langle \Omega, \ldots, \Omega \rangle$, and is chain-closed.

The analog of Theorem 3 now follows easily be application of the general Theorems of § 4.4.1; it says that:

"Every monotonically structured system of recursive definitions

$\bar{F}(\bar{x}) \Leftarrow \bar{\tau}(\bar{F}, \bar{x})$ has a monotonic least strong fixpoint, $\hat{\bar{f}}_{\tau}$.
In addition:

$$\hat{\bar{f}}_{\tau} = \text{lub}\{\bar{\tau}^{i}(\bar{\Omega}) \mid i \geq 0\}$$

The analog of Theorem 4, concerning the least weak fixpoint $\hat{\bar{f}}_{\tau}$, is also true.

E. Fixpoint Computations for Systems of Recursive Definitions.

Finally, the fixpoint computations which we have seen in Chapter 5 can be easily extended to compute the least fixpoints $\hat{\bar{f}}_{\tau}$ and $\hat{\bar{f}}_{\sigma}$ of a monotonically structured system of recursive definitions.

The standard simplifications need no modification. Also, if α is a monotonically structured term free of $\bar{x}$ such that $\tilde{\alpha}(\bar{\Omega}, \ldots) \equiv a$ for some $a \neq \Omega$, then $\text{Simp}(\alpha) = \underset{\sim}{a}$. (Analog of the last lemma of § 5.2)

The full substitutions are easily extended to terms with several function variables: the inductive definition of $\text{Fsubst}_{\bar{\tau}}(\alpha)$ will be identical to the one in § 5.3.1 except for case (v), which becomes:

"if $\alpha = F_j(\alpha_1, \ldots, \alpha_n)$, for some j, $1 \leq j \leq k$, then:

$$\text{Fsubst}_{\bar{\tau}}(\alpha) = \tau_j(\bar{F}, \text{Fsubst}_{\bar{\tau}}(\alpha_1), \ldots, \text{Fsubst}_{\bar{\tau}}(\alpha_n))."$$

The important lemma of § 5.3.1 has its analog here, which can be expressed, for example, in the following way:

"For any correct term α and any k-tuple $\bar{\tau}$ of compatible terms,

for any $\bar{f} \in [pf_n(D^+)]^k$, $\bar{\bar{F}} \in (D^+)^n$:

$$\widetilde{\text{Fsubst}}_{\bar{\bar{F}}}(\alpha)\ (\bar{f}, \bar{\bar{F}}) = \widetilde{\alpha}\ (\widetilde{\bar{\bar{F}}}\ [\bar{f}], \bar{\bar{F}})."$$

The <u>full computation</u> of a term $\alpha(\bar{\bar{F}},\bar{x})$ for $\bar{x} = \bar{c} \in (D^+)^n$ using a monotonically structured system of recursive definitions $\bar{F}(\bar{x}) \Leftarrow \bar{\bar{\tau}}(\bar{F},\bar{x})$ will be a sequence of terms α_i such that:

$$\alpha_0 = \alpha(\bar{\bar{F}},\underset{\sim}{c}) ,$$

and, for every $p \geq 0$:

$$\alpha_{2p+1} = \text{Simp}\ (\alpha_{2p}) ,$$

$$\alpha_{2p+2} = \text{Fsubst}_{\bar{\bar{F}}}(\alpha_{2p+1}) .$$

These terms have the property (analogous to the first lemma of § 5.3.3) that:

" $\forall \bar{f} \in [pf_n(D^+)]^k$, $\forall p \geq 0$:

$$\widetilde{\alpha}_{2p}(\bar{f}, \ ) = \widetilde{\alpha}_{2p+1}(\bar{f}, \ ) = \widetilde{\alpha}_0(\widetilde{\bar{\bar{\tau}}}^p[\bar{f}], \ ) ".$$

Then the analog of the second lemma of § 5.3.3 also holds. It says:

"Let α be a monotonically structured correct term, $\bar{F}(\bar{x}) \Leftarrow \bar{\tau}(\bar{F}, \bar{x})$ a monotonically structured system of recursive definitions, and let $\bar{c} \in (D^+)^n$. Then, the full computation of α for $\bar{x} = \bar{c}$ using the system terminates with $\underset{\sim}{a}$ iff $\widetilde{\alpha}(\hat{\bar{f}}_{\tau},\bar{c}) = a$; it does not terminate iff $\widetilde{\alpha}(\hat{\bar{f}}_{\tau},\bar{c}) \equiv \perp$. ($\hat{\bar{f}}_{\tau}$ is the least strong fixpoint of the system)."

The computed function $\bar{f}_c$ obtained by full computation of the

system $\bar{F}(\bar{x}) \Leftarrow \bar{\tau}(\bar{F},\bar{x})$ will be the k-tuple $\langle f_{c,1}, \ldots, f_{c,k} \rangle$, where, for every j, $1 \leq j \leq k$, $f_{c,j} \in pf_n(D^+)$ is the partial function obtained by the full computation of $F_j(\bar{x})$ using the system. Then the analog of Theorem 2 says that $\bar{f}_c = \hat{\bar{\tau}}$, i.e. the computed function obtained by full computation of a monotonically structured system of recursive definitions is the least strong fixpoint of the system.

The definitions and fixpoint properties concerning innermost computations carry over to systems of recursive definitions in a similar fashion:

Parallel Innermost Substitutions are readily extended by changing 5.4.1 (v) to:

"$\alpha = F_j(\alpha_1, \ldots, \alpha_n)$ for some j, $1 \leq j \leq k$;

if, for every i, $1 \leq i \leq n$, $\alpha_i = \underset{\sim}{a}_i$, $\underset{\sim}{a}_i \in C$, then:

$Psubst_{\bar{\tau}}(\alpha) = \tau_j(F, \underset{\sim}{a}_1, \ldots, \underset{\sim}{a}_n)$,

otherwise:

$Psubst_{\bar{\tau}}(\alpha) = F(Psubst_{\bar{\tau}}(\alpha_1), \ldots, Psubst_{\bar{\tau}}(\alpha_n))$ ".

The definition of the standard innermost computation is likewise readily extended.

The computed function $\bar{f}_s$ obtained by the standard innermost computation of the system $\bar{F}(\bar{x}) \Leftarrow \bar{\tau}(\bar{F}, \bar{x})$ will be the k-tuple $\langle f_{s,1}, \ldots, f_{s,k} \rangle$, where, for every j, $1 \leq j \leq k$, $f_{s,j} \in pf_n(D)$ is obtained by the standard innermost computation of $F_j(\bar{x})$ using the system.

Then we have the analogous of the lemma of § 5.4.3 and of Theorem 6. The latter will read, for example:

"the computed function obtained by standard innermost computation

of a monotonically structured system of recursive definitions is the least weak fixpoint of the system."

Likewise, the generalization of safe innermost computations can be done with no special difficulty. The definitions of the safe simplifications and safe innermost substitutions are unchanged, since they depend only on the given functions.

The analog of Theorem 7 is again true and reads: "Any computed function obtained by safe innermost computation of a monotonically structured system of recursive definitions is the least weak fixpoint of the system".

BIBLIOGRAPHY

Bekić (1969): Hans Bekić, "Definable Operations in General Algebras, and the Theory of Automata and Flowcharts". Unpublished Memo, IBM Laboratory, Vienna. (December 1969).

Burstall (1969): R. M. Burstall, "Proving Properties of Programs by Structural Induction". Computer Journal, Vol. 12, pp. 41-48 (1969).

Cohn (1965): P. M. Cohn, "Universal Algebras", Harper and Row, London. (1965).

Gödel (1934): K. Gödel, "On Undecidable Propositions of Formal Mathematical Systems". Institute for Advanced Study, Princeton, New Jersey (1934).

Halmos (1960): Paul R. Halmos, "Naive Set Theory", D. van Nostrand, Princeton, New Jersey (1960).

Herbrand (1931): Jacques Herbrand, "Sur la non-contradiction de l'Arithmétique". J. reine angew. Math. Vol. 166, pp. 1-8, (1931).

Kleene (1936): S. C. Kleene, "General Recursive Functions of Natural Numbers", Math. Ann., Vol. 112, pp. 727-742, (1936).

Kleene (1952): S. C. Kleene, "Introduction to Metamathematics", D. Van Nostrand, Princeton, New Jersey (1952).

Manna and McCarthy (1970): Zohar Manna and John McCarthy, "Properties of Programs and Partial Function Logic", in Machine Intelligence 5, Edinburgh University Press (1970).

Manna and Pnueli (1970): Zohar Manna and Amir Pnueli, "Formalization of Properties of Functional Programs", JACM, Vol. 17, No. 3 pp. 555-569 (July 1970).

McCarthy (1963): John McCarthy, "A Basis for a Mathematical Theory of Computation", in Computer Programming and Formal Systems, pp. 33-70 (eds. P. Braffort and D. Hirschberg). Amsterdam, North Holland (1963). Also in Proceedings of the Western Joint Computer Conference, pp. 225-238. New York, Spartan Books (1961).

Morris (1968): James H. Morris, "Lambda-Calculus Models of Programming Languages", Ph.D. Thesis, Project MAC, M.I.T. (MAC-TR-57) (December 1968).

Park (1969): David Park, "Fixpoint Induction and Proofs of Program Properties", in Machine Intelligence 5, Edinburgh University Press (1970).

Park (1970): David Park, "Notes on a Formalism for reasoning about Schemas", unpublished notes, University of Warwick (November 1970).

Rosen (1971): B. K. Rosen, "Tree Manipulating Systems and Church-Rosser Theorems", Internal Report, Engineering Sciences Laboratory, Harvard University, Cambridge, Mass. (1971). [A preliminary version of this paper appeared in the Proceedings of the Second Annual Symposium on Theory of Computing, Northampton, Mass., May 1970].

Scott (1969): D. Scott, "A Type Theoretical Alternative to CUCH, ISWIM, OWHY". Unpublished notes, Oxford University (1969).

Scott (1970): D. Scott, "An outline of a Mathematical Theory of Computation", Oxford University Computing Laboratory, Programming Research Group, Technical Monograph PRG 2 (November 1970).

Suppes (1960): Patrick Suppes, "Axiomatic Set Theory", D. Van Nostrand, Princeton, New Jersey (1960).

Tarski (1955): A. Tarski, " A Lattice-Theoretical fixpoint theorem and its Applications". Pacific J. of Mathematics, Vol. 5, pp. 285-309 (1955).

Vuillemin (1972): J. Vuillemin, "Proof Techniques for Recursive Programs", Ph.D. Thesis, Computer Science Department, Stanford University (to appear).